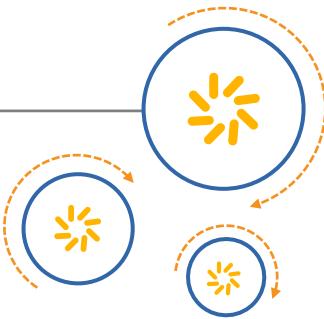




Qualcomm Technologies, Inc.



# Wireless LAN Access Point (Driver Version 10.4)

## Programmer Guide

80-Y8053-1 Rev. YV

October 30, 2017

QUALCOMM®  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

**Confidential and Proprietary – Qualcomm Technologies, Inc.**

**NO PUBLIC DISCLOSURE PERMITTED:** Please report postings of this document on public servers or websites to:  
[DocCtrlAgent@qualcomm.com](mailto:DocCtrlAgent@qualcomm.com).

**Restricted Distribution:** Not to be distributed to anyone who is not an employee of either Qualcomm Technologies, Inc. or its affiliated companies without the express approval of Qualcomm Configuration Management.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc.

Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

Qualcomm Technologies, Inc.  
5775 Morehouse Drive  
San Diego, CA 92121  
U.S.A.

© 2014-2017 Qualcomm Technologies, Inc. All rights reserved

## Revision history

Revision	Date	Description
A	June 2014	Initial release for AP 10.4 software. For 10.2 releases of AP software, refer to the Programmer's Guide 80-Y7207-1.
B	August 2014	<p>Updates to:</p> <ul style="list-style-type: none"> <li>■ Section 3.5.1, "Options Usage"</li> <li>■ Section 4.4.1.5, "Driver Security Services"</li> <li>■ Section 4.4.2.6, "Driver Security Services"</li> <li>■ Section 5.3, "Spectral scan and analysis"</li> <li>■ Section 5.13.3, "Driver Files"</li> <li>■ Section 5.16.7.3, "WRAP Design Target and Limitation"</li> </ul> <p>New sections:</p> <ul style="list-style-type: none"> <li>■ Section 5.31, "TCP Segmentation Offload"</li> <li>■ Section 5.32, "Large Receive Offload"</li> <li>■ Section 5.33, "Scatter/Gather DMA"</li> <li>■ Section 5.34, "Checksum Offload"</li> <li>■ Section 5.16.7.4, "Q-WRAP L2 Bridging"</li> <li>■ Section 5.16.7.5, "Data Flow"</li> <li>■ Section 7.11, "Raw Mode Tx/Rx"</li> </ul>
C	October 2014	<p>Updates to:</p> <ul style="list-style-type: none"> <li>■ Table 3-1, "Build Options"</li> <li>■ Section 5.11.3.1, "Reserve dedicated descriptors for VI clients" (and sub-sections)</li> <li>■ Section 5.11.3.3.1, "Settings"</li> <li>■ Section 5.11.3.4.2, "Selection of penalization client"</li> <li>■ Section 5.13.5.2.1, "Configuration Descriptions"</li> <li>■ Section 5.13.5.2.2, "Configuration Examples"</li> <li>■ Section 5.15.7.3, "WRAP Design Target and Limitation"</li> <li>■ Section 5.15.7.5.5, "Pseudocode"</li> <li>■ Section 5.15.7.8, "DBDC Q-WRAP"</li> <li>■ Section 7.5.3, "Example Call Flow in HOST Driver"</li> <li>■ Section 7.5.4, "Supported LOWI Message Types"</li> <li>■ Section 7.6.1.1, "Interference Detection Mechanism "</li> <li>■ Figure 7-11, "Overview of Network-based Positioning AP Software System Components"</li> </ul> <p>New sections:</p> <ul style="list-style-type: none"> <li>■ Section 5.35, "Airtime Fairness"</li> <li>■ Section 7.12, "Thermal Mitigation"</li> <li>■ Section 7.13, "Single AP Band Steering"</li> <li>■ Section 7.18, "Wake on Wireless AP Assist"</li> <li>■ Section 7.19, "QCaché"</li> </ul>

Revision	Date	Description
D	November 2014	Updates to: ■ <a href="#">Figure 5-26, "Channel Hopping Algorithm"</a> New sections: ■ <a href="#">Section 7.20, "Host Tx Flow Control"</a>
E	January 2015	Added <a href="#">Section 7.21, "Descriptor Configuration"</a>
F	March 2015	Added <a href="#">Section 5.35.4 "ATF design for direct attach architecture"</a>
G	March 2015	Minor changes to formatting, no change to document
H	May 2015	Added the following: ■ <a href="#">Section 4.11</a> ■ <a href="#">Section 7.16</a> Updated the following: ■ <a href="#">Section 5.10.1.5</a>
J	July 2015	Updated the following: ■ <a href="#">Section 1.1</a> ■ <a href="#">Section 1.1.1</a> ■ <a href="#">Section 1.1.2</a> ■ <a href="#">Section 1.2.11.2</a>
K	July 2015	Added the following sections: ■ <a href="#">Section 5.37</a>
L	July 2015	Updated <a href="#">Chapter 7</a>
M	July 2015	Updated <a href="#">Chapter 7</a>
N	September 2015	Added the following sections: ■ <a href="#">Section 4.5.3.1</a> ■ <a href="#">Section 5.39</a>
P	September 2015	Added the following sections: ■ <a href="#">Section 5.37</a> ■ <a href="#">Section 7.23</a> ■ <a href="#">Section 7.24</a>

Revision	Date	Description
R	October 2015	<p>Updated the following sections:</p> <ul style="list-style-type: none"> <li>■ <a href="#">Section 4.11</a></li> <li>■ <a href="#">Section 4.11.1.6</a></li> <li>■ <a href="#">Section 4.11.2</a></li> <li>■ <a href="#">Section 4.11.2.1</a></li> <li>■ <a href="#">Section 4.11.2.2</a></li> <li>■ <a href="#">Section 5.3.2.3.2</a></li> <li>■ <a href="#">Section 5.3.3.3.1</a></li> <li>■ <a href="#">Section 5.3.4</a></li> <li>■ <a href="#">Section 7.22</a></li> </ul> <p>Updated the following tables:</p> <ul style="list-style-type: none"> <li>■ <a href="#">Table 4-5</a></li> <li>■ <a href="#">Table 5-7</a></li> <li>■ <a href="#">Table 5-9</a></li> <li>■ <a href="#">Table 5-10</a></li> <li>■ <a href="#">Table 5-14</a></li> </ul> <p>Added the following tables:</p> <ul style="list-style-type: none"> <li>■ <a href="#">Table 4-6</a></li> <li>■ <a href="#">Table 4-8</a></li> <li>■ <a href="#">Table 4-10</a></li> </ul>
T	December 2015	<p>Added the following sections:</p> <ul style="list-style-type: none"> <li>■ <a href="#">Section 5.35.5</a></li> <li>■ <a href="#">Section 5.35.6</a></li> <li>■ <a href="#">Section 5.35.7</a></li> <li>■ <a href="#">Section 5.35.8</a></li> </ul> <p>Updated <a href="#">Section 5.39</a></p>
U	January 2016	Updated <a href="#">Chapter 5</a> and <a href="#">Chapter 7</a>
V	February 2016	<ul style="list-style-type: none"> <li>■ Updated the following sections: <ul style="list-style-type: none"> <li>□ <a href="#">Section 4.11.1.13</a></li> <li>□ <a href="#">Section 5.15.7.8</a></li> </ul> </li> <li>■ Added following sections: <ul style="list-style-type: none"> <li>□ <a href="#">Section 4.11.3</a></li> <li>□ <a href="#">Section 5.15.7.11</a></li> <li>□ <a href="#">Section 5.40</a></li> <li>□ <a href="#">Section 7.25</a></li> <li>□ <a href="#">Section 7.26</a></li> </ul> </li> </ul>
W	March 2016	<ul style="list-style-type: none"> <li>■ Added the following sections: <ul style="list-style-type: none"> <li>□ <a href="#">Section 4.5.3.2</a></li> <li>□ <a href="#">Section 5.35.9</a></li> <li>□ <a href="#">Section 5.41</a></li> </ul> </li> <li>■ Updated the following sections: <ul style="list-style-type: none"> <li>□ <a href="#">Section 5.35.7</a></li> <li>□ <a href="#">Section 5.35.8</a></li> </ul> </li> </ul>

Revision	Date	Description
Y	March 2016	<ul style="list-style-type: none"> <li>■ Added the following sections: <ul style="list-style-type: none"> <li>□ <a href="#">Section 5.35.10</a></li> <li>□ <a href="#">Section 7.27</a></li> </ul> </li> <li>■ Updated <a href="#">Section 5.16</a></li> </ul>
YA	May 2016	<ul style="list-style-type: none"> <li>■ Added the following sections: <ul style="list-style-type: none"> <li>□ <a href="#">Section 1.3</a></li> <li>□ <a href="#">Section 5.16.4</a></li> <li>□ <a href="#">Section 5.42</a></li> <li>□ <a href="#">Section 7.13.3.4</a></li> <li>□ <a href="#">Section 7.28</a></li> </ul> </li> <li>■ Updated the following: <ul style="list-style-type: none"> <li>□ <a href="#">Section 3.1</a></li> <li>□ <a href="#">Section 4.5.3.2</a></li> <li>□ <a href="#">Section 4.11.1.5</a></li> <li>□ <a href="#">Section 4.11.3</a></li> <li>□ <a href="#">Section 5.7.2</a></li> <li>□ <a href="#">Section 5.16.2</a></li> <li>□ <a href="#">Section 5.35.11</a></li> <li>□ <a href="#">Section 7.3.1.2.4</a></li> <li>□ <a href="#">Section 7.13.7</a></li> <li>□ <a href="#">Section 7.14.1</a></li> </ul> </li> </ul>
YB	June 2016	<ul style="list-style-type: none"> <li>■ Added the following sections: <ul style="list-style-type: none"> <li>□ <a href="#">Section 4.5.3.3</a></li> <li>□ <a href="#">Section 5.24.5</a></li> <li>□ <a href="#">Section 5.43</a></li> <li>□ <a href="#">Section 5.44</a></li> <li>□ <a href="#">Section 5.45</a></li> <li>□ <a href="#">Section 7.17</a></li> <li>□ <a href="#">Section 7.29</a></li> </ul> </li> <li>■ Updated the following: <ul style="list-style-type: none"> <li>□ <a href="#">Table 7-21</a></li> </ul> </li> </ul>
YC	July 2016	<ul style="list-style-type: none"> <li>■ Updated the following: <ul style="list-style-type: none"> <li>□ <a href="#">Section 4.5.3.3</a></li> <li>□ <a href="#">Figure 4-8</a></li> <li>□ <a href="#">Section 5.18.2.1</a></li> <li>□ <a href="#">Section 7.5</a></li> <li>□ <a href="#">Section 7.13.6</a></li> <li>□ <a href="#">Section 7.17.3</a></li> <li>□ <a href="#">Section 7.21</a></li> </ul> </li> </ul>
YD	August 2016	<ul style="list-style-type: none"> <li>■ Updated <a href="#">Section 5.61</a></li> </ul>

Revision	Date	Description
YE	August 2016	<ul style="list-style-type: none"> <li>■ Updated the following           <ul style="list-style-type: none"> <li>□ <a href="#">Section 3.4.5.9</a></li> <li>□ <a href="#">Section 3.7.2.1</a></li> <li>□ <a href="#">Section 3.7.3.2</a></li> <li>□ <a href="#">Section 5.8.1</a></li> <li>□ <a href="#">Section 5.8.2</a></li> <li>□ <a href="#">Table 7-21</a></li> <li>□ <a href="#">Table 7-11</a></li> <li>□ <a href="#">Section 7.21.3.1</a></li> <li>□ <a href="#">Section 7.21.3.2</a></li> </ul> </li> <li>■ Added the following           <ul style="list-style-type: none"> <li>□ <a href="#">Section 7.20.1.3</a></li> <li>□ <a href="#">Section 7.20.1.4</a></li> <li>□ <a href="#">Section 7.20.1.5</a></li> </ul> </li> </ul>
YF	September 2016	<ul style="list-style-type: none"> <li>■ Updated the following:           <ul style="list-style-type: none"> <li>□ <a href="#">Section 7.13.9</a></li> <li>□ <a href="#">Section 7.20.1</a></li> <li>□ <a href="#">Table 7-14</a></li> <li>□ <a href="#">Table 7-18</a></li> <li>□ <a href="#">Table 7-21</a></li> </ul> </li> </ul>
YG	October 2016	<p>Following sections are added/updated for <a href="#">QCA_Networking_2016.SPF.4.0 ES release</a></p> <ul style="list-style-type: none"> <li>■ Added the following:           <ul style="list-style-type: none"> <li>□ <a href="#">Section 5.17</a></li> <li>□ <a href="#">Table 5-19</a></li> <li>□ <a href="#">Table 5-20</a></li> <li>□ <a href="#">Section 7.43</a></li> <li>□ <a href="#">Section 7.45</a></li> </ul> </li> <li>■ Updated the following:           <ul style="list-style-type: none"> <li>□ Acronyms table in preface</li> <li>□ <a href="#">Section 1.2.8</a></li> <li>□ <a href="#">Section 1.3</a></li> <li>□ <a href="#">Section 2.5.3.1</a></li> <li>□ <a href="#">Section 3.4.2</a></li> <li>□ <a href="#">Section 4.10.4.2</a></li> <li>□ <a href="#">Section 5.3.4.3.1</a></li> <li>□ <a href="#">Section 5.23</a></li> <li>□ <a href="#">Section 5.30.1</a></li> <li>□ <a href="#">Section 5.35.1</a></li> <li>□ <a href="#">Section 5.37.1</a></li> <li>□ <a href="#">Section 7.35</a></li> <li>□ <a href="#">Figure 1-3</a></li> <li>□ <a href="#">Table 3-1</a></li> <li>□ <a href="#">Section 5.16.7.7</a></li> <li>□ <a href="#">Section 5.51.4</a></li> </ul> </li> </ul>

Revision	Date	Description
YH	October 2016	<p>Following section is added for <b>IPQ4019.ILQ.1.2.2 ED release</b></p> <ul style="list-style-type: none"> <li>■ <a href="#">Section 7.46</a></li> </ul>
YJ	November 2016	<p>Following sections are added/updated for <b>QCA_Networking_2016.SPF.4.0 FC release</b></p> <ul style="list-style-type: none"> <li>■ Added the following: <ul style="list-style-type: none"> <li>□ <a href="#">Section 1.4</a>, <a href="#">Section 1.5</a>, <a href="#">Section 1.6</a></li> <li>□ <a href="#">Section 4.5.3.4</a></li> <li>□ <a href="#">Section 4.6</a>, <a href="#">Section 4.7</a>, <a href="#">Section 4.11</a>, <a href="#">Section 4.12</a></li> <li>□ <a href="#">Section 5.3.3</a></li> <li>□ <a href="#">Section 5.7</a></li> <li>□ <a href="#">Section 5.27</a>, <a href="#">Section 5.28</a></li> <li>□ <a href="#">Section 5.40</a>, <a href="#">Section 5.42</a>, <a href="#">Section 5.43</a>, <a href="#">Section 5.44</a>, <a href="#">Section 5.45</a></li> <li>□ <a href="#">Section 5.49</a></li> <li>□ <a href="#">Section 5.52</a></li> <li>□ <a href="#">Section 5.59</a></li> <li>□ <a href="#">Section 5.62</a>, <a href="#">Section 5.63</a>, <a href="#">Section 5.64</a></li> <li>□ <a href="#">Section 6.3</a>, <a href="#">Section 6.4</a></li> <li>□ <a href="#">Section 7.15</a>, <a href="#">Section 7.16</a></li> <li>□ <a href="#">Section 7.24</a></li> <li>□ <a href="#">Section 7.36</a></li> <li>□ <a href="#">Section 7.44</a></li> <li>□ <a href="#">Section 7.47</a>, <a href="#">Section 7.48</a>, <a href="#">Section 7.49</a></li> </ul> </li> <li>■ Updated the following: <ul style="list-style-type: none"> <li>□ <a href="#">Section 1.3</a></li> <li>□ <a href="#">Section 3.4.5.9</a></li> <li>□ <a href="#">Section 4.5.3.2</a></li> <li>□ <a href="#">Section 5.16.7.7</a></li> <li>□ <a href="#">Section 5.51.3</a></li> <li>□ <a href="#">Section 7.17.1.6</a></li> <li>□ <a href="#">Table 7-18</a></li> <li>□ <a href="#">Section 7.43</a></li> </ul> </li> </ul>

Revision	Date	Description
YK	December 2016	<p>Following sections are added/updated for <b>QCA_Networking_2016.SPF.4.0 CS release</b></p> <ul style="list-style-type: none"><li>■ Added the following:<ul style="list-style-type: none"><li>□ <a href="#">Section 5.53</a></li><li>□ <a href="#">Section 5.57.5</a></li><li>□ <a href="#">Section 5.60</a></li><li>□ <a href="#">Section 7.13.2.5</a></li><li>□ <a href="#">Section 7.17.1.8</a></li><li>□ <a href="#">Section 7.19</a></li><li>□ <a href="#">Section 7.22</a></li><li>□ <a href="#">Section 7.25</a></li><li>□ <a href="#">Section 7.26</a></li><li>□ <a href="#">Section 7.27</a></li><li>□ <a href="#">Section 7.28</a></li><li>□ <a href="#">Section 7.40</a></li></ul></li><li>■ Updated the following:<ul style="list-style-type: none"><li>□ <a href="#">Section 1.1.2</a></li><li>□ <a href="#">Section 1.2.11.1</a></li><li>□ <a href="#">Section 1.3</a></li><li>□ <a href="#">Section 1.3.2</a></li><li>□ <a href="#">Section 1.3.3</a></li><li>□ <a href="#">Section 5.51.4</a></li><li>□ <a href="#">Section 7.17.3</a></li><li>□ <a href="#">Section 7.20</a></li><li>□ <a href="#">Section 7.39</a></li></ul></li></ul>

Revision	Date	Description
YL	January 2017	<p>The following sections are added/updated for <b>QCA_Networking_2016.SPF.4.0 CSU1 release</b></p> <ul style="list-style-type: none"> <li>■ Added the following: <ul style="list-style-type: none"> <li>□ Section 2.6, "Improvements to WMI layer architecture"</li> <li>□ Section 2.7, "Transport layer enhancements for modularization"</li> <li>□ Section 2.8, "Plugin of Data Path Interface"</li> <li>□ Section 2.9, "Optimization of OS abstraction layer"</li> <li>□ Section 6.7, "Customize full-offload configuration for peak KPI"</li> <li>□ Section 16.8, "Wi-Fi SON API for lbd blacklist"</li> <li>□ Section 19.12, "Wi-Fi SON: Channel planning on 2.4 and 5 GHz bands for distributed Wi-Fi systems"</li> <li>□ Section 19.13, "Wi-Fi SON: Ethernet roaming per port"</li> <li>□ Section 19.15, "Wi-Fi SON with PLC: Support for daisy-chain and star topologies"</li> <li>□ Section 20.5, "Coexistence of Bluetooth and ZigBee with WLAN on IPQ8064"</li> </ul> </li> <li>■ Updated the following: <ul style="list-style-type: none"> <li>□ Section 17.7.1, "Implementation of Tx and Rx data flows for DBDC repeater"</li> <li>□ Section 17.7.2, "Configuration of Tx and Rx data flows for DBDC repeater"</li> <li>□ Section 19.4, "Wi-Fi SON: Full Daisy Chain support across multiple hops"</li> <li>□ Section 19.5, "Wi-Fi SON: Support across multiple Hy-Fi bridges"</li> <li>□ Section 19.6, "Save STA information to flash for Wi-Fi SON"</li> <li>□ Section 19.7, "Detection of WAN and LAN sample scenarios"</li> <li>□ Section 19.8, "Network loop prevention sample scenarios"</li> </ul> </li> </ul>

Revision	Date	Description
YM	February 2017	<p>The following sections are added/updated for <b>QCA_Networking_2017.SPF.5.0 ES release:</b></p> <ul style="list-style-type: none"> <li>■ Added the following: <ul style="list-style-type: none"> <li>□ <a href="#">Section 3.5, "Restore WLAN configuration after a firmware assert"</a></li> <li>□ <a href="#">Section 3.6, "Replace the direct calls to printk API with QDF print "</a></li> <li>□ <a href="#">Section 4.3.3.4, "ACFG maximum clients event "</a></li> <li>□ <a href="#">Section 5.12, "Support for dynamic LAN and WAN for single net device "</a></li> <li>□ <a href="#">Section 6.8, "IPQ806x IP Crypto Offload"</a></li> <li>□ <a href="#">Section 6.9, "IPsec Tunnels"</a></li> <li>□ <a href="#">Section 6.10, "IPsec Acceleration"</a></li> <li>□ <a href="#">Section 6.11, "IPsec Bring Up"</a></li> <li>□ <a href="#">Section 6.12, "Extended NSS signaling"</a></li> <li>□ <a href="#">Section 6.13, "Full-offload configuration for QCA955x"</a></li> <li>□ <a href="#">Section 10.11, "ATF strict scheduling per SSID or virtual device for QCA9984, IPQ4019, and QCA9886"</a></li> <li>□ <a href="#">Section 11.4, "Configure AMPDU per VAP for DA and OL modes"</a></li> <li>□ <a href="#">Section 12.5, "HNAT on QCA9531, QCA9558, and QCA9563 chipsets"</a></li> <li>□ <a href="#">Section 13.11, "Wi-Fi Alliance MBO functions"</a></li> <li>□ <a href="#">Section 13.15, "Enhanced dynamic beacons"</a></li> <li>□ <a href="#">Section 13.19, "Customized neighbor report"</a></li> <li>□ <a href="#">Section 14.13, "Inclusion of channel or band capability preference in STA association requests"</a></li> <li>□ <a href="#">Section 14.14, "Transmission of QCN IE in beacons, probe responses, and association responses"</a></li> <li>□ <a href="#">Section 14.15, "Cloud-based channel change"</a></li> </ul> </li> <li>■ Updated the following: <ul style="list-style-type: none"> <li>□ <a href="#">Section 6.7, "Customize full-offload configuration for peak KPI"</a></li> <li>□ <a href="#">Section 13.14, "Dynamic beacon"</a></li> <li>□ <a href="#">Table 19-5, "Parameters for LED indications"</a></li> </ul> </li> </ul>

Revision	Date	Description
YN	March 2017	<p>The following sections are added/updated for <b>QCA_Networking_2017.SPF.5.0 ED release</b> and <b>QCA_Networking_2017.SPF.5.2 ED2 release</b>:</p> <ul style="list-style-type: none"> <li>■ Added the following: <ul style="list-style-type: none"> <li>□ <a href="#">Section 3.7, "Receive flow steering"</a></li> <li>□ <a href="#">Section 5.5, "Enhanced scan results IOCTL to display neighbor AP details "</a></li> <li>□ <a href="#">Section 5.13, "TDMA method"</a></li> <li>□ <a href="#">Section 8.7, "Modify Tx chain-mask without reconnecting clients"</a></li> <li>□ <a href="#">Section 9.4, "Wi-Fi LTE-U operation"</a></li> <li>□ <a href="#">Section 9.5, "TxOPs of 10 ms for best effort access category"</a></li> <li>□ <a href="#">Section 10.5, "Disable ANI to control packet detection threshold"</a></li> <li>□ <a href="#">Section 11.6, "Configure AMSDU per VAP for DA and OL modes"</a></li> <li>□ <a href="#">Section 13.20, "Mirror Tx packets from AP VAP to a monitor VAP in direct attach radios"</a></li> <li>□ <a href="#">Section 13.21, "Tx data packet captures in OL mode"</a></li> <li>□ <a href="#">Section 13.22, "Transmission of MBO Reason Code attribute in BTM Request frames"</a></li> <li>□ <a href="#">Section 13.23, "Support offload-based radios to connect to Wi-Fi hotspot of phones"</a></li> <li>□ <a href="#">Section 13.24, "QCN broadcast probe response"</a></li> <li>□ <a href="#">Section 14.16, "Scanning RSSI of NACs"</a></li> <li>□ <a href="#">Section 14.17, "Channel-ranking information in ACS reports"</a></li> <li>□ <a href="#">Section 15.7, "Spectral debug enhancements"</a></li> <li>□ <a href="#">Section 16.7, "Band steering in QWRAP mode"</a></li> <li>□ <a href="#">Section 16.8, "Configure band steering per SSID pair"</a></li> <li>□ <a href="#">Section 16.9, "AP steering for legacy clients"</a></li> <li>□ <a href="#">Section 16.10, "Display the hyd and lbd versions"</a></li> <li>□ <a href="#">Section 19.16, "Wi-Fi SON: Determine the join and leave events of peers"</a></li> <li>□ <a href="#">Section 19.17, "Wi-Fi SON: Multinode WPS support for push button events"</a></li> <li>□ <a href="#">Section 21.2, "802.11ad firmware recovery"</a></li> <li>□ <a href="#">Section 21.3, "802.11ad: Fast session transfer"</a></li> <li>□ <a href="#">Section 21.4, "Massive array for 802.11ad"</a></li> <li>□ <a href="#">Section 22.7, "Generate target core dump"</a></li> <li>□ <a href="#">Section 22.8, "Enhanced client statistics display for RSSI, Tx/Rx packets, and STBC"</a></li> <li>□ <a href="#">Section 22.9, "Connection state logging functionality"</a></li> <li>□ <a href="#">Section 22.10, "Smart logging method"</a></li> <li>□ <a href="#">Section 22.11, "CE logging functionality"</a></li> <li>□ <a href="#">Section 22.12, "Buffer tracking"</a></li> <li>□ <a href="#">Section 22.13, "Packet log and Tx completion message correlation to mirror Tx data packets on QCA9880"</a></li> </ul> </li> <li>■ Updated the following: <ul style="list-style-type: none"> <li>□ <a href="#">Section 22.2.6.3, "Required CLIs to test the feature"</a></li> </ul> </li> </ul>

Revision	Date	Description
YP	April 2017	<p>The following sections are added/updated for <b>QCA_Networking_2017.SPF.5.0 CS release</b> and <b>QCA_Networking_2017.SPF.5.2 ED3 release</b>:</p> <ul style="list-style-type: none"> <li>■ Added the following: <ul style="list-style-type: none"> <li>□ Section 6.14, "Save system memory content for troubleshooting NSS problems"</li> <li>□ Section 7.5, "Configure short guard interval for all MCSs" on page 401</li> <li>□ Section 19.18, "Wi-Fi SON: Cloning of backhaul VAP credentials for multi-SSID with traffic separation"</li> </ul> </li> <li>■ Updated the following: <ul style="list-style-type: none"> <li>□ Section 14.17, "Channel-ranking information in ACS reports"</li> <li>□ Section 16.2.7.2, "Load Balancing Daemon"</li> <li>□ Section 16.8, "Configure band steering per SSID pair"</li> <li>□ Section 16.9, "AP steering for legacy clients"</li> <li>□ Section 20, "Coexistence of WLAN with NoDS mesh, BT, ZigBee, and CSRmesh"</li> </ul> </li> </ul>
YR	June 21, 2017	<p>The following sections are added/updated for <b>QCA_Networking_2017.SPF.5.0.2 CS release</b> and <b>QCA_Networking_2017.SPF.5.2 ED4 release</b>:</p> <ul style="list-style-type: none"> <li>■ Added the following: <ul style="list-style-type: none"> <li>□ Section 3.8, "Early detection of bit corruption during firmware assert"</li> <li>□ Section 8.8, "Disable buffering of multicast frames for OL and DA mode stations in power-save mode"</li> <li>□ Section 8.9, "Support for the same Tx power-out in dBm with antenna across all chains "</li> <li>□ Section 8.10, "Transmit power control per SSID for control frames"</li> <li>□ Section 10.14, "ATF hierarchy distribution"</li> <li>□ Section 10.27, "FILS capability"</li> <li>□ Section 13.6.1, "Radiotap changes for PCI device ID and antenna DB in sniffer feedback"</li> <li>□ Section 13.25, "Configure Tx Ack timeout value"</li> <li>□ Section 19.21, "Best uplink algorithm for PLC interface metrics in a daisy chain Wi-Fi SON and PLC network"</li> <li>□ Section 19.22, "Interoperation of different PLC chipset models in a Wi-Fi SON network"</li> <li>□ Section 22.14, "Enhanced client statistics display for Tx management frames, minimum/maximum RSSI, and received data MPDUs"</li> <li>□ Section 22.15, "Integration of Wi-Fi statistics with vendors for extended statistics"</li> <li>□ Section 22.16, "Display of decoded watchdog timer signature for QCA9980"</li> <li>□ Section 22.17, "Smart logging for HALPHY modules"</li> <li>□ Section 22.18, "WMI support for debug statements in UTF"</li> </ul> </li> <li>■ Updated the following: <ul style="list-style-type: none"> <li>□ Section 8.3, "Transmit Power Control (TPC)"</li> <li>□ Section 8.5, "Advanced Enterprise for 10.4"</li> <li>□ Section 19.14, "Wi-Fi SON: PLC backhaul"</li> </ul> </li> </ul>
YT	June 23, 2017	Updated the Section 17.4.4, "MAC Address Translation".

Revision	Date	Description
YU	July 2017	<p>The following sections are added/updated for <b>QCA_Networking_2017.SPF.5.0.2 CSU1.1 release</b> and <b>QCA_Networking_2017.SPF.5.2 ED5 release</b>:</p> <ul style="list-style-type: none"> <li>■ Added the following: <ul style="list-style-type: none"> <li>□ Section 5.14, "Multi-credentials for VAP"</li> <li>□ Section 6.16, "Multiple WAN"</li> <li>□ Section 6.21, "VLAN traffic separation over Wi-Fi"</li> <li>□ Section 6.24, "802.3x flow control"</li> <li>□ Section 6.26, "ACL-based steering"</li> <li>□ Section 6.27, "Multiple PPPoE sessions"</li> <li>□ Section 10.6.8, "Configure SSID grouping for ATF"</li> <li>□ Section 10.28, "MU-MIMO"</li> <li>□ Section 13.26, "UCI commands to configure RADIUS parameters for retries "</li> <li>□ Section 13.27, "WAPI with 16 VAP clients"</li> <li>□ Section 13.28, "Configure random number generation mode"</li> <li>□ Section 14.18, "Send channel statistics every second for DCS using ACFG event logs"</li> <li>□ Section 14.19, "Channel blocking "</li> <li>□ Section 17.11, "Automatic addition of Ethernet client to ProxySTA in QWRAP mode"</li> <li>□ Section 22.19, "Display latency for each PPDU using packet log tool"</li> <li>□ Section 22.20, "Enhanced TID queue statistics"</li> </ul> </li> <li>■ Updated the following: <ul style="list-style-type: none"> <li>□ Section 16.9, "AP steering for legacy clients"</li> </ul> </li> </ul>

Revision	Date	Description
YV	October 2017	<p>The following sections are added/updated for <b>QCA_Networking_2017.SPF.5.0.3 release</b> and <b>QCA_Networking_2017.SPF.5.3 release</b>:</p> <ul style="list-style-type: none"> <li>■ Added the following: <ul style="list-style-type: none"> <li>□ <a href="#">Section 6.12.1, "Configuration commands for extended NSS signaling"</a></li> <li>□ <a href="#">Section 6.12.2, "Limitations for extended NSS signaling"</a></li> <li>□ <a href="#">Section 9.6, "ETSI compliance for COT and CAT on QCA9980 and QCA9880"</a></li> <li>□ <a href="#">Section 9.7, "5.9 GHz frequency band "</a></li> <li>□ <a href="#">Section 9.8, "ETSI 5 GHz rules in Europe (ETSI EN 301 893)"</a></li> <li>□ <a href="#">Section 10.30, "FILS Discovery frames"</a></li> <li>□ <a href="#">Section 10.31, "Support for dynamically enabling and disabling burst"</a></li> <li>□ <a href="#">Section 11.7, "Using SA query timeout for PMF"</a></li> <li>□ <a href="#">Section 11.8, "Retrieve and display radar channel details"</a></li> <li>□ <a href="#">Section 16.9.3, "Enhanced AP steering of 802.11k unfriendly clients as legacy clients"</a></li> <li>□ <a href="#">Section 16.9.4, "AP steering for legacy clients on QCA9886 chipsets"</a></li> <li>□ <a href="#">Section 16.11, "Soft-blocking for safe connections"</a></li> <li>□ <a href="#">Section 17.4.7, "Q-WRAP (Qualcomm Wireless Repeater AP)"</a></li> <li>□ <a href="#">Section 17.12, "Enhanced Independent Repeater scan algorithm"</a></li> <li>□ <a href="#">Section 17.13, "Support for same SSID on root and repeater APs in star topology in QWRAP"</a></li> <li>□ <a href="#">Section 19.1.1.4, "Guidelines for Wi-Fi SON on QCA9531 platforms"</a></li> <li>□ <a href="#">Section 19.23, "Wi-Fi SON: Skip PFS and improve airtime calculation"</a></li> <li>□ <a href="#">Section 19.24, "Support for REH172 as a root AP"</a></li> <li>□ <a href="#">Section 19.25, "Wi-Fi SON: Diagnostics and troubleshooting"</a></li> <li>□ <a href="#">Section 22.21, "Enhanced client statistics for acknowledgment failures, modulation rates, authentication details "</a></li> </ul> </li> <li>■ Updated the following: <ul style="list-style-type: none"> <li>□ <a href="#">Section 2.5.6, "umac"</a></li> <li>□ <a href="#">Section 4.1.1, "Usage of build options"</a></li> <li>□ <a href="#">Section 12.2.1, "Multicast Enhancement (ME)"</a></li> <li>□ <a href="#">Section 16.9, "AP steering for legacy clients"</a></li> <li>□ <a href="#">Section 19.3, "Wi-Fi SON: Multi-SSID and Traffic Separation"</a></li> </ul> </li> </ul>

# Contents

<b>1 WLAN AP Driver Architecture</b>	54
1.1 Wireless LAN	54
1.1.1 Direct Attach Architecture	55
1.1.2 Full Offload Architecture	55
1.2 WLAN Software Architecture	57
1.2.1 Hardware Abstraction Layer (HAL)	57
1.2.2 HAL API	58
1.2.3 Lower MAC (LMAC)	58
1.2.4 LMAC API	59
1.2.5 Upper MAC (UMAC)	59
1.2.6 WLAN API	59
1.2.7 OS Interface Layer (OSIF)	59
1.2.8 Qualcomm Driver Framework (QDF)	59
1.2.9 Atheros Service Framework (ASF)	60
1.2.10 WLAN Application	60
1.2.11 Unified Software WLAN Architecture	60
1.3 WLAN driver modules architecture until QCA_Networking_2016.SPF.2.0 release	63
1.3.1 Common WLAN Driver Modules	64
1.3.2 Direct-Attach specific WLAN Driver Modules	64
1.3.3 Offload specific WLAN Driver Modules	64
1.4 Offload and direct-attach driver modularization	65
1.5 Additional changes for WLAN driver modularization	70
1.6 Wi-Fi Calibration Data mapping	76
1.6.1 Calibration data stored in flash	76
1.6.2 Pre-initialization scripts	76
1.6.3 Wi-Fi scripts	77
<b>2 WLAN AP Driver Layers</b>	78
2.1 Major data structures	78
2.1.1 OSIF data structures	79
2.1.2 UMAC data structures	80
2.1.3 LMAC data structures	81
2.1.4 HAL data structures	81
2.2 Directory structure	82
2.3 Common data structures	82
2.3.1 vap	82

2.3.2	node .....	82
2.3.3	ic .....	82
2.3.4	sc .....	82
2.3.5	ath_hal .....	82
2.4	Driver layers .....	83
2.4.1	OS shim layer .....	83
2.4.2	802.11 protocol layer (net, UMAC) .....	83
2.4.3	Device layer (ath, LMAC) .....	84
2.4.4	HAL .....	85
2.5	Source tree layout .....	86
2.5.1	asf .....	86
2.5.2	qdf .....	86
2.5.3	htc .....	87
2.5.4	include .....	87
2.5.5	lmac .....	87
2.5.6	umac .....	88
2.6	Improvements to WMI layer architecture .....	91
2.6.1	WMI layer design .....	92
2.6.2	WMI layer interactions with other modules .....	93
2.6.3	WMI initialization sequence .....	95
2.6.4	WMI backward compatibility .....	96
2.6.5	Cookie or pool-based allocation for HTC packets .....	96
2.6.6	WMI directory structure .....	96
2.7	Transport layer enhancements for modularization .....	97
2.7.1	Host architecture .....	97
2.7.2	Host Low Level Design .....	98
2.7.3	HIF/CE Major data structures .....	100
2.7.4	Concurrent support for USB/SIDO/PCI .....	102
2.7.5	Register table support for different chips .....	103
2.7.6	NSS WLAN offload support .....	104
2.8	Plugin of Data Path Interface .....	105
2.8.1	Host architecture .....	105
2.8.2	Dependency on other modules .....	106
2.8.3	Host low-level design .....	106
2.8.4	Driver-specific APIs .....	111
2.8.5	Changes to non-public interfaces .....	112
2.8.6	Changes to public interfaces .....	112
2.9	Optimization of OS abstraction layer .....	115
2.9.1	ADF to QDF transition architecture .....	115
2.9.2	List of ADF and QDF APIs .....	116
2.10	Deprecated Qualcomm driver features .....	122

<b>3 WLAN AP Driver Operations</b>	131
3.1 Processing Contexts and Synchronization	131
3.1.1 ISR Processing	131
3.1.2 Softirq/Tasklet Processing	131
3.1.3 Process Context	132
3.1.4 Synchronization	132
3.2 Buffer Management	132
3.2.1 WBUF abstraction	132
3.2.2 Descriptors Abstractions	133
3.2.3 Receive Buffer Management	134
3.2.4 Transmit Buffer Management	135
3.3 Queue Management	135
3.3.1 Hardware Queues	135
3.3.2 LMAC Queuing Support	135
3.4 Code Flow	138
3.4.1 Configuration Path	138
3.4.2 Data Path	145
3.4.3 Data Path — Partial Offload	153
3.4.4 OL Data Structures	163
3.5 Restore WLAN configuration after a firmware assert	164
3.5.1 Radio and VAP restoration	165
3.5.2 DFS operations	165
3.5.3 Configuration recovery list	166
3.5.4 Event notifications	166
3.6 Replace the direct calls to printk API with QDF print	166
3.7 Receive flow steering	167
3.7.1 RPS and RFS in Linux kernel	168
3.7.2 Steering for packets from ESS to WLAN	169
3.7.3 Interworking of components for RFS	169
3.7.4 GMAC driver	170
3.7.5 RFS module	171
3.7.6 RFS daemon	174
3.7.7 SSDK module APIs	175
3.7.8 Sample configuration	175
3.8 Early detection of bit corruption during firmware assert	177
3.8.1 Target recovery parameters	179
<b>4 WLAN AP Driver Build and Configuration Methods</b>	180
4.1 Build Options	180
4.1.1 Usage of build options	180
4.2 Runtime configuration options	187
4.2.1 wlanconfig Utility	187
4.2.2 Wireless Tools	188

4.3	Configuration API (ACFG) .....	190
4.3.1	Theory of Operation .....	190
4.3.2	Implementation .....	192
4.3.3	Configurations .....	198
<b>5</b>	<b>WLAN AP Modes</b> .....	205
5.1	AP Mode .....	205
5.1.1	Terminology .....	205
5.1.2	Implementation .....	205
5.2	AP-only Mode .....	210
5.2.1	Data Path Optimizations For Improved CPU Utilization .....	210
5.3	Client Mode .....	214
5.3.1	Terminology .....	214
5.3.2	Implementation .....	214
5.3.3	Configuration .....	218
5.4	AP and Client Security .....	218
5.4.1	Theory of Operation .....	218
5.4.2	Driver Interface for Hostap Implementation .....	221
5.4.3	Configurations .....	226
5.5	Enhanced scan results IOCTL to display neighbor AP details .....	227
5.6	Multiple BSSID .....	229
5.6.1	Implementation .....	229
5.6.2	VAP management .....	230
5.6.3	Beacon generation .....	230
5.6.4	Configure an AP interface .....	231
5.7	VLAN Support .....	231
5.7.1	Overview .....	231
5.7.2	Terminology .....	231
5.7.3	Theory of Operation .....	232
5.7.4	VLAN implementation .....	232
5.7.5	Configuration .....	234
5.8	Hy-Fi WLAN .....	234
5.8.1	API .....	237
5.8.2	Configuration .....	238
5.9	Location wireless interface (LOWI) .....	239
5.9.1	LOWI client side library .....	240
5.9.2	LOWI server process .....	243
5.9.3	Add a LOWI client on the AP .....	246
5.9.4	Client side library classes .....	248
5.9.5	Configure round-trip time on AP using UCI commands .....	291
5.10	Static MAC ID generation .....	291
5.11	Positioning: LOWI on AP .....	292
5.11.1	RTT Types .....	293

5.11.2	How to Perform Ranging .....	294
5.11.3	Example Call Flow in HOST Driver .....	296
5.11.4	Supported LOWI Message Types .....	297
5.12	Support for dynamic LAN and WAN for single net device .....	298
5.12.1	LAN/WAN group compositions .....	298
5.13	TDMA method .....	302
5.13.1	Sample configuration scenario .....	302
5.13.2	Limitations .....	305
5.13.3	Sequence of frames exchange .....	305
5.14	Multi-credentials for VAP .....	307
<b>6</b>	<b>Acceleration and Offload Features .....</b>	<b>308</b>
6.1	TCP Segmentation Offload .....	308
6.1.1	Implementation .....	308
6.1.2	Configuration .....	308
6.2	Large Receive Offload .....	309
6.2.1	Implementation .....	309
6.2.2	Configuration .....	309
6.3	Checksum Offload .....	309
6.3.1	Implementation .....	310
6.4	Offload Optimized Host Data Path .....	310
6.5	NSS Wi-Fi Offload .....	313
6.5.1	Features .....	314
6.5.2	Host Driver Split Architecture .....	314
6.5.3	Standard Profile .....	315
6.5.4	Enterprise Profile .....	316
6.6	HNAT Wi-Fi Offload .....	316
6.6.1	QCA8327 .....	316
6.6.2	Hardware NAT Accelerator .....	316
6.6.3	Configuration .....	318
6.7	Customize full-offload configuration for peak KPI .....	318
6.7.1	Load the full-offload modules .....	319
6.7.2	Modify the full-offload network configuration for flash mode .....	320
6.7.3	Modify the Ethernet MTU configuration .....	321
6.7.4	Reduce CPU utilizations .....	322
6.7.5	Disable kernel prints .....	322
6.7.6	Sigma DUT configurations .....	322
6.7.7	Bypass-LAN configuration .....	322
6.7.8	Limitation .....	323
6.8	IPQ806x IP Crypto Offload .....	323
6.8.1	Driver .....	323
6.8.2	Offload .....	324
6.8.3	Crypto Offload APIs .....	325

6.8.4	Supported Crypto Algorithms . . . . .	326
6.8.5	Restrictions . . . . .	328
6.9	IPsec Tunnels . . . . .	329
6.9.1	Campus deployment . . . . .	329
6.9.2	Remote branch office . . . . .	330
6.9.3	IPsec with crypto offload . . . . .	331
6.10	IPsec Acceleration . . . . .	331
6.10.1	Acceleration model . . . . .	332
6.10.2	APIs . . . . .	333
6.10.3	Restrictions . . . . .	335
6.11	IPsec Bring Up . . . . .	335
6.11.1	IPsec setup . . . . .	336
6.11.2	MTU . . . . .	338
6.11.3	IPsec Bring up (LAN to WAN) . . . . .	338
6.11.4	IPsec bring up (WLAN to WAN) . . . . .	341
6.11.5	Verify NSS IPsec features . . . . .	343
6.11.6	AES-192 software fallback . . . . .	343
6.11.7	AES-CTR . . . . .	344
6.11.8	Initiator . . . . .	344
6.11.9	Responder . . . . .	346
6.11.10	Statistics . . . . .	347
6.12	Extended NSS signaling . . . . .	349
6.12.1	Configuration commands for extended NSS signaling . . . . .	354
6.12.2	Limitations for extended NSS signaling . . . . .	355
6.12.3	Sample configuration scenario for extended NSS signaling . . . . .	355
6.13	Full-offload configuration for QCA955x . . . . .	356
6.13.1	Flash boot mode . . . . .	356
6.13.2	MII boot . . . . .	357
6.13.3	Reworks for host AP136-0xx for host mode . . . . .	362
6.14	Save system memory content for troubleshooting NSS problems . . . . .	365
6.15	Known networking acceleration limitations . . . . .	366
6.16	Multiple WAN . . . . .	367
6.16.1	UCI configuration of the multiwan . . . . .	367
6.16.2	Sample multiwan connection . . . . .	369
6.16.3	Analyze the multiwan . . . . .	370
6.16.4	Sample multiwan tests . . . . .	371
6.16.5	Limitations . . . . .	371
6.17	Multipath support in ECM . . . . .	372
6.17.1	Single path implementation . . . . .	372
6.17.2	Multipath model . . . . .	372
6.17.3	Multipath Test Topology . . . . .	373
6.18	IPv6 MAP-T . . . . .	374
6.18.1	MAP-T topology . . . . .	374

6.18.2	Design changes .....	375
6.19	IPv6 NAPT .....	377
6.19.1	Detailed information on tunneling with IPv6 .....	378
6.19.2	Verify NSS acceleration .....	379
6.19.3	Configuration commands .....	379
6.19.4	MAP-T Configuration .....	380
6.20	IPv6 processing .....	386
6.20.1	IPv4 to IPv6 transition on WAN link .....	387
6.21	VLAN traffic separation over Wi-Fi .....	391
6.21.1	Sample configuration scenarios .....	391
6.22	Link aggregation group .....	393
6.22.1	LAG requirements .....	394
6.22.2	LAG architecture .....	395
6.22.3	HLOS updates .....	395
6.22.4	Rule push to NSS .....	395
6.22.5	Destruction of rules .....	396
6.22.6	Link-down event .....	397
6.22.7	Link-up event .....	397
6.22.8	NSS Sync Notifications .....	397
6.22.9	LAG interface up/down .....	397
6.22.10	Design .....	397
6.22.11	Feature Test Setup – TCP/UDP .....	399
6.22.12	Linux Bonding Driver Options .....	399
6.22.13	Determine the type of egress interface .....	400
6.23	LACP framework .....	400
6.23.1	User interface of link aggregation .....	403
6.23.2	Status of link aggregation .....	405
6.23.3	User space link aggregation daemon .....	406
6.23.4	Framework of LACPD daemon .....	407
6.23.5	Data structure of LACPD daemon .....	409
6.23.6	Receive and transmit LACP packet .....	410
6.23.7	Trunk configuration .....	413
6.23.8	Requirement for hardware switch about port state .....	413
6.23.9	Link status notification .....	413
6.23.10	Failover in link aggregation .....	414
6.23.11	Integration with other subsystem .....	414
6.23.12	Limitations and Notice .....	414
6.24	802.3x flow control .....	415
6.24.1	Requirements .....	415
6.24.2	Implementation .....	415
6.24.3	Configure 802.3x flow control .....	416
6.25	LAN port ID handling NSS and IPQ40x8 platforms .....	417
6.25.1	EDMA Multiple VLANs .....	418

6.25.2	Sample Topology .....	419
6.25.3	2 VLANs (1+4) The default case .....	419
6.25.4	5 VLANs (1+1+1+1+1) .....	421
6.25.5	2 VLANs (3 + 2) .....	424
6.25.6	3 VLANs (2 + 2 + 1) .....	426
6.25.7	Commands to check SFE counters .....	428
6.25.8	Sample output of ECM dump .....	428
6.25.9	Sample output of SFE dump .....	432
6.25.10	Routing and multiflow configuration commands .....	432
6.25.11	Commands to disable and enable SFE .....	439
6.25.12	HNAT commands .....	439
6.25.13	HNAT counters .....	439
6.25.14	ECM counters .....	439
6.26	ACL-based steering .....	440
6.26.1	ESS switch ACL feature .....	440
6.26.2	GRE flow steering use case .....	442
6.26.3	ARP entry with load balance .....	443
6.26.4	Interface entry with MAC .....	444
6.26.5	GRE tunnel identification .....	444
6.26.6	UCI configuration .....	445
6.26.7	Limitations .....	453
6.26.8	Configuration example .....	453
6.27	Multiple PPPoE sessions .....	458
6.27.1	Start a PPPoE server .....	459
6.27.2	Stop a PPPoE server .....	460
6.27.3	Sample configuration .....	460
6.27.4	Create multiple PPPoE sessions .....	461
6.27.5	Create Dual WAN port .....	462
6.27.6	Display PPPoE statistics .....	462
6.27.7	UCI commands .....	463
6.27.8	Add routes .....	466
<b>7</b>	<b>Rate Control Features .....</b>	<b>468</b>
7.1	Rate Control .....	468
7.1.1	Theory of Operation .....	468
7.1.2	Direct Attach Implementation .....	469
7.1.3	Offload Implementation .....	475
7.1.4	Code .....	481
7.1.5	API .....	481
7.1.6	Configuration .....	482
7.1.7	Rate Control for Special Traffic Types .....	483
7.2	256 QAM rate support in 2.4 GHz operation .....	483
7.2.1	256 QAM Rate Support Negotiation .....	483

7.2.2	Implementation .....	484
7.2.3	CLI Commands .....	484
7.3	Disable Selected MCS For Given SSID .....	484
7.3.1	Theory of Operation .....	485
7.3.2	Implementation .....	485
7.3.3	Offload Implementation .....	486
7.3.4	CLI Guide .....	486
7.3.5	Validation .....	487
7.4	Disable selected legacy rates for given SSID .....	488
7.4.1	Purpose .....	488
7.4.2	Overview .....	488
7.4.3	Background theory .....	488
7.4.4	Implementation .....	489
7.4.5	Configuration .....	490
7.4.6	Validation .....	490
7.4.7	Limitation .....	491
7.5	Configure short guard interval for all MCSs .....	491
<b>8</b>	<b>Power Management Techniques .....</b>	<b>492</b>
8.1	UAPSD WMM Power Save (WMM-PS) .....	492
8.1.1	Terminology .....	492
8.1.2	Theory of Operation .....	493
8.1.3	WMM Power-Save Advertisement .....	493
8.1.4	U-APSD Enabled STA Association .....	493
8.1.5	STA and Power Management .....	494
8.1.6	STA and UAPSD Trigger .....	494
8.1.7	Implementation .....	494
8.2	Power Management .....	498
8.2.1	Terminology .....	498
8.2.2	Theory of Operation .....	498
8.2.3	Direct-attach Implementation .....	499
8.2.4	Offload Implementation .....	502
8.3	Transmit Power Control (TPC) .....	504
8.3.1	Theory of Operation .....	505
8.3.2	Implementation .....	505
8.3.3	Additional feature .....	512
8.4	Configure transmit power for management frames per SSID .....	513
8.5	Advanced Enterprise for 10.4 .....	517
8.5.1	Advanced Enterprise API Support .....	517
8.6	Thermal Mitigation .....	522
8.6.1	Objectives .....	522
8.6.2	Interfaces .....	522
8.6.3	WLAN Driver and Firmware .....	523

8.6.4	Thermal tool .....	524
8.6.5	Implementation .....	524
8.6.6	TMD-WLAN Interface .....	527
8.7	Modify Tx chain-mask without reconnecting clients .....	528
8.7.1	Sample configuration scenario .....	528
8.7.2	Known limitations .....	531
8.8	Disable buffering of multicast frames for OL and DA mode stations in power-save mode ..	531
8.9	Support for the same Tx power-out in dBm with antenna across all chains .....	533
8.10	Transmit power control per SSID for control frames .....	536
<b>9</b>	<b>Regulatory Compliance of WLAN APs .....</b>	<b>538</b>
9.1	Regulatory Compliance .....	538
9.1.1	Operation .....	538
9.1.2	Direct Attach Implementation .....	540
9.1.3	Software APIs .....	541
9.1.4	Offload Regulatory Compliance .....	542
9.2	Offload FCC regulatory rules to firmware .....	543
9.3	Preserve regulatory settings after HLOS replacement .....	549
9.4	Wi-Fi LTE-U operation .....	549
9.4.1	LTE-U requirements .....	550
9.4.2	WLAN driver design .....	552
9.4.3	LTE-U configuration .....	563
9.4.4	Display statistics and packet logs .....	567
9.4.5	Known limitations .....	573
9.5	TxOPs of 10 ms for best effort access category .....	573
9.5.1	Configure double backoff for 10 ms TxOPs .....	574
9.5.2	Sample configuration scenario .....	575
9.6	ETSI compliance for COT and CAT on QCA9980 and QCA9880 .....	579
9.6.1	CAL and COT considerations .....	579
9.6.2	Design methodology .....	581
9.6.3	Test methods .....	581
9.6.4	Limitations .....	581
9.7	5.9 GHz frequency band .....	581
9.7.1	Design overview .....	582
9.7.2	Verify the 5.9 GHz band .....	586
9.8	ETSI 5 GHz rules in Europe (ETSI EN 301 893) .....	588
<b>10</b>	<b>Memory and Performance .....</b>	<b>599</b>
10.1	ART2 Calibration .....	599
10.1.1	Theory of Operation .....	599
10.2	Interrupt Mitigation/Moderation .....	604
10.2.1	Interrupts .....	604
10.2.2	Mitigation: A Simple Implementation .....	604

10.2.3	Mitigation: Unintended Consequence .....	605
10.2.4	MAC Mitigation Support .....	605
10.3	SIFS Burst .....	605
10.3.1	Theory of Operation .....	606
10.3.2	Implementation .....	606
10.4	Adaptive Noise Immunity (ANI) .....	606
10.4.1	Theory of Operation .....	607
10.4.2	Implementation .....	607
10.5	Disable ANI to control packet detection threshold .....	612
10.6	Airtime fairness .....	612
10.6.1	ATF requirement .....	612
10.6.2	ATF design for offload architecture .....	613
10.6.3	ATF design for direct attach architecture .....	618
10.6.4	ATF – OBSS Interference .....	625
10.6.5	ATF – Tx Buffer distribution .....	626
10.6.6	ATF—Supported maximum clients in direct attach .....	626
10.6.7	ATF—SSID grouping support in direct attach .....	626
10.6.8	Configure SSID grouping for ATF .....	626
10.6.9	ATF—Dynamic enable/disable feature for direct attach architecture .....	627
10.6.10	ATF —Guaranteed Throughput or Bandwidth Fairness .....	627
10.6.11	ATF—Per-SSID scheduling policy in Direct Attach .....	628
10.7	Airtime configuration GUI .....	630
10.8	ATF event logging .....	635
10.9	ATF fair-scheduling based on WMM access classes .....	636
10.10	ATF bandwidth fairness or minimum guaranteed throughput .....	636
10.11	ATF strict-scheduling per SSID or virtual device for QCA9880 .....	637
10.12	ATF strict scheduling per SSID or virtual device for QCA9984, IPQ4019, and QCA9886 .....	638
10.13	ATF support for MU-MIMO .....	639
10.14	ATF hierarchy distribution .....	639
10.14.1	Guidelines for ATF hierarchy distribution .....	640
10.15	Scatter/Gather DMA .....	641
10.15.1	Implementation .....	641
10.15.2	Configuration .....	641
10.16	Coordinated ATF between root AP and repeater .....	642
10.17	Peer flow control data path .....	642
10.17.1	Packet flow .....	643
10.18	Content-aware routing .....	643
10.19	Override MU-MIMO capability based on vendor OUI .....	643
10.20	QCache .....	649
10.21	Firmware code sign and firmware authentication at host .....	650
10.21.1	Firmware file header format .....	650
10.21.2	Firmware sign validation .....	654

10.21.3	API specification .....	655
10.22	WLAN LED implementation .....	656
10.22.1	Direct attach implementation for WLAN LED control .....	656
10.22.2	Offload path implantation for WLAN LED control .....	656
10.23	Pre-allocation of the required runtime memory .....	657
10.23.1	Memory pool data structures .....	657
10.23.2	Initialization of memory pools .....	658
10.23.3	Runtime allocation and free interfaces .....	658
10.24	Memory Footprint Reduction on QCA9880/QCA9886/QCA9889 .....	659
10.25	RDK-B HAL API PHASE 2 .....	664
10.25.1	wifi_getApAssociatedDevicesHighWatermarkThreshold .....	664
10.25.2	wifi_setApAssociatedDevicesHighWatermarkThreshold .....	664
10.25.3	wifi_getApAssociatedDevicesHighWatermarkThresholdReached .....	664
10.25.4	wifi_getApAssociatedDevicesHighWatermark .....	664
10.25.5	wifi_getApAssociatedDevicesHighWatermarkDate .....	664
10.25.6	wifi_setRadioTrafficStatsMeasure .....	665
10.25.7	wifi_getRadioStatsReceivedSignalLevel .....	665
10.26	Wi-Fi reload .....	665
10.26.1	Avoiding module reload during wifi up/down .....	665
10.26.2	Code changes .....	666
10.26.3	Reloading the modules with new module parameters .....	666
10.27	FILS capability .....	667
10.27.1	Wi-Fi driver enhancements for FILS support .....	667
10.27.2	Hostapd enhancements for FILS support .....	668
10.28	MU-MIMO .....	668
10.28.1	MU-MIMO per VAP statistics .....	670
10.28.2	Tx sounding statistics .....	670
10.28.3	Tx MU statistics .....	670
10.28.4	Tx Selfgen statistics .....	671
10.28.5	Tx transmitted MU packet count per peer .....	672
10.28.6	User positions in different MU groups for a particular AID .....	672
10.28.7	Sample test scenario .....	672
10.28.8	MU-MIMO data path enhancements overview .....	682
10.28.9	Enhanced MU-MIMO debugging statistics .....	686
10.28.10	Enhanced MU-MIMO for scoring metrics and MU grouping .....	688
10.29	Latency optimization .....	689
10.29.1	Problem statement .....	689
10.29.2	Objectives .....	690
10.29.3	Design .....	691
10.30	FILS Discovery frames .....	692
10.30.1	Enable/disable FILS Discovery Frame support .....	693
10.30.2	UCI commands to configure FILS Discovery Frames .....	694
10.30.3	WMI command to enable FD frames .....	694

10.30.4	SWFDA event .....	694
10.30.5	Send FD buffer frame to the firmware .....	695
10.31	Support for dynamically enabling and disabling burst .....	695
<b>11</b>	<b>IEEE 802.11 Features .....</b>	<b>696</b>
11.1	802.11ac (Very High Throughput) Features .....	696
11.1.1	Initialization .....	697
11.1.2	Configuration .....	698
11.1.3	Operation .....	698
11.1.4	AP Mode .....	699
11.1.5	STA Mode .....	699
11.2	160/80+80 MHz Operation .....	700
11.2.1	Allowed Channels .....	700
11.2.2	NSS negotiation in 160/80+80MHz modes .....	708
11.2.3	Revised 160/80+80 MHz operation signaling .....	712
11.3	802.11n Features .....	714
11.3.1	A-MPDU Aggregation (Peer Feature Negotiation) .....	714
11.3.2	Data Transmit Aggregation .....	715
11.4	Configure AMPDU per VAP for DA and OL modes .....	716
11.4.1	AMPDU per VAP .....	718
11.5	802.11w Protected Management Frames (PMF) .....	718
11.5.1	Theory of Operation .....	719
11.5.2	Implementation .....	719
11.5.3	Driver Files .....	719
11.5.4	hostapd Files .....	720
11.5.5	Data Receive Aggregation .....	723
11.5.6	A-MSDU Aggregation .....	723
11.6	Configure AMSDU per VAP for DA and OL modes .....	724
11.7	Using SA query timeout for PMF .....	725
11.7.1	Protected management frames (PMF) parameters .....	726
11.7.2	UCI command to configure SA query timeout for PMF .....	726
11.8	Retrieve and display radar channel details .....	727
11.8.1	Sample configuration scenario .....	727
<b>12</b>	<b>QoS and Policy Control Operations .....</b>	<b>729</b>
12.1	Access Control List .....	729
12.1.1	Implementation .....	729
12.1.2	Configuration .....	730
12.2	iQue (Intelligent QoS for User Experience) .....	730
12.2.1	Multicast Enhancement (ME) .....	730
12.2.2	Head of the Line Blocking Removal (HBR) .....	735
12.3	Support for SFE on IPQ8064 platforms—Coexistence between offload engines .....	736
12.4	Reject clients with low SNR .....	739
12.5	HNAT on QCA9531, QCA9558, and QCA9563 chipsets .....	739

12.5.1	ARP packet process .....	741
12.5.2	Static NAT implementation .....	742
12.5.3	NAPT implementation .....	744
12.5.4	Sample configuration scenario of static and dynamic NAT .....	749
<b>13</b>	<b>Beacons and Frames Transmission .....</b>	<b>751</b>
13.1	Software Retry (SWR) .....	751
13.1.1	Theory of Operation .....	752
13.1.2	Implementation .....	754
13.1.3	Configuration .....	758
13.2	Bursted Beaoning .....	758
13.3	Wake on Wireless AP Assist .....	760
13.3.1	High Level Design .....	760
13.3.2	WoW packet format on Wire Shark .....	761
13.4	Dynamic Transmit Chainmask Selection (DTCS) .....	761
13.4.1	Theory of Operation .....	761
13.4.2	Implementation .....	762
13.4.3	Configuration .....	762
13.5	Quick STA Kickout (QSK) .....	762
13.5.1	Theory of Operation .....	762
13.5.2	Implementation .....	763
13.5.3	Configuration .....	763
13.6	Support for radiotap header .....	763
13.6.1	Radiotap changes for PCI device ID and antenna DB in sniffer feedback ..	766
13.7	OCE Overview .....	767
13.8	HotSpot 2.0 .....	780
13.8.1	Beacon/Probe Response IE Processing .....	780
13.8.2	Probe Request Processing .....	781
13.8.3	Action Frame Processing .....	781
13.8.4	Proxy ARP .....	781
13.8.5	DGAF Disable .....	782
13.8.6	L2TIF .....	782
13.9	Hidden SSID support in strict passive scan .....	783
13.10	Multiband Operation (MBO) overview .....	785
13.11	Wi-Fi Alliance MBO functions .....	789
13.11.1	Multiband operation architecture .....	790
13.11.2	Multiband operation topology .....	790
13.11.3	Multiband operation protocol .....	794
13.11.4	Multiband Operation IE Attributes and Frame Formats .....	801
13.12	Host Tx flow control .....	813
13.13	Descriptor configuration for FW .....	813
13.14	Dynamic beacon .....	814
13.15	Enhanced dynamic beacons .....	817

13.16 Beacon rate control per SSID for offload mode . . . . .	820
13.17 TX99 Mode . . . . .	820
13.17.1 TX99 for 11n Radio . . . . .	821
13.17.2 TX99 ioctl Option . . . . .	822
13.17.3 TX99 Command Guide . . . . .	824
13.17.4 TX99 for 802.11AC Radio . . . . .	826
13.17.5 athtestcmd Tx Rate Table . . . . .	831
13.17.6 Testing with tx99tool and athtestcmd . . . . .	835
13.18 Raw Mode Tx/Rx . . . . .	837
13.18.1 Introduction . . . . .	837
13.18.2 Scope . . . . .	838
13.18.3 Design . . . . .	841
13.19 Customized neighbor report . . . . .	846
13.20 Mirror Tx packets from AP VAP to a monitor VAP in direct attach radios . . . . .	847
13.21 Tx data packet captures in OL mode . . . . .	848
13.22 Transmission of MBO Reason Code attribute in BTM Request frames . . . . .	849
13.22.1 Implementation of MBO Reason Code . . . . .	851
13.23 Support offload-based radios to connect to Wi-Fi hotspot of phones . . . . .	854
13.24 QCN broadcast probe response . . . . .	855
13.24.1 Requirements addressed with QCN broadcast probe response . . . . .	855
13.24.2 Guidelines for QCN broadcast probe response . . . . .	855
13.24.3 Design changes for QCN broadcast probe response . . . . .	855
13.24.4 Configure QCN broadcast probe response capability . . . . .	857
13.24.5 Sample configuration scenario . . . . .	858
13.25 Configure Tx Ack timeout value . . . . .	858
13.25.1 Configure and verify the Tx Ack timeout value . . . . .	858
13.25.2 Verify that ACK timeout value is not modified after channel change . . . . .	859
13.26 UCI commands to configure RADIUS parameters for retries . . . . .	860
13.27 WAPI with 16 VAP clients . . . . .	861
13.27.1 Sample configuration scenarios . . . . .	864
13.28 Configure random number generation mode . . . . .	872
<b>14 Channel Selection . . . . .</b>	<b>874</b>
14.1 Channel Width Management (CWM 20/40) . . . . .	874
14.1.1 Theory of Operation . . . . .	874
14.1.2 Implementation . . . . .	875
14.1.3 API . . . . .	878
14.1.4 Configuration . . . . .	878
14.1.5 Offload implementation . . . . .	879
14.2 Dynamic Frequency Selection (DFS) . . . . .	880
14.2.1 Terminology . . . . .	880
14.2.2 Theory of Operation . . . . .	881
14.2.3 Implementation . . . . .	882

14.2.4 Configuration .....	901
14.2.5 CLI Commands .....	901
14.2.6 Troubleshooting .....	902
14.3 Zero wait for DFS channels .....	902
14.4 ETSI pre-CAC Host driver and firmware enhancements .....	905
14.5 EACS PLUS: ACS/OBSS .....	912
14.5.1 Requirements .....	912
14.5.2 Terminology .....	913
14.5.3 EACS_PLUS vs. EACS Major Metric Correction .....	913
14.5.4 EACS plus algorithm changes .....	916
14.5.5 Back ground ACS/OBSS scan .....	920
14.5.6 OBSS check .....	920
14.6 Intelligent Channel Manager .....	921
14.6.1 Design .....	922
14.6.2 ICM Channel Selection Logic .....	924
14.6.3 Interface with Dynamic Channel Selection .....	931
14.7 UMAC Auto Channel Selection (Scanning) .....	932
14.7.1 Source of Interference .....	932
14.7.2 Selection of Channels .....	932
14.7.3 Configuration .....	933
14.7.4 Dynamic Channel Selection (DCS) .....	933
14.7.5 Channel Loading and Channel Hopping .....	933
14.7.6 Channel Block List .....	937
14.8 Non-intrusive channel selection .....	938
14.9 80 MHz Scan .....	938
14.10 Dynamic Channel Selection-Interference Mitigation (DCS-IM) for 802.11n & 802.11a .....	940
14.10.1 Implementation .....	940
14.10.2 Configuration .....	944
14.11 Modify channel width in CSA for 5 GHz .....	946
14.12 Enhanced channel switch and channel scan API .....	946
14.12.1 User space application and kernel space driver communication mechanism .....	947
14.12.2 New IOCTL Commands implemented by WLAN driver .....	947
14.12.3 NETLINK broadcast event notification sent by WLAN driver .....	952
14.12.4 Limitations .....	954
14.13 Inclusion of channel or band capability preference in STA association requests .....	954
14.14 Transmission of QCN IE in beacons, probe responses, and association responses .....	955
14.15 Cloud-based channel change .....	958
14.15.1 Guidelines for cloud-based channel change functionality .....	959
14.15.2 Flow diagram of repeater AP behavior .....	960
14.15.3 AP VAP behavior during cloud-based channel change .....	961
14.15.4 STA VAP behavior during cloud-based channel change .....	962
14.15.5 Limitations with cloud-based channel change .....	962
14.15.6 Sample configuration scenarios .....	962

14.16 Scanning RSSI of NACs .....	965
14.17 Channel-ranking information in ACS reports .....	967
14.17.1 Method of calculation of channel score .....	968
14.17.2 Configure and display ACS channel-ranking .....	969
14.18 Send channel statistics every second for DCS using ACFG event logs .....	971
14.19 Channel blocking .....	972
14.19.1 Guidelines .....	972
14.19.2 Specify the block list .....	972
14.19.3 Specify the blocking mode .....	972
14.19.4 Block list .....	973
<b>15 Spectral Scan and Analysis .....</b>	976
15.1 Generation I spectral scan .....	977
15.1.1 Spectral scan parameters .....	979
15.1.2 FFT report format .....	980
15.2 Generation II spectral scans .....	981
15.2.1 Generation II enhancements .....	981
15.2.2 Known Limitations .....	985
15.2.3 QCA999x/998x Spectral Scan Event Reporting .....	985
15.2.4 Generation II Spectral Scan Configuration Registers .....	990
15.3 Software Architecture .....	992
15.3.1 Spectral Host Driver .....	992
15.3.2 Spectral Indirection Table Mapping .....	994
15.3.3 Generation II Spectral Scan Firmware Functionality Overview .....	998
15.3.4 Generation II Spectral Scan Host-Firmware Interaction .....	998
15.4 Classification .....	1001
15.4.1 Microwave Oven Detection .....	1001
15.4.2 Frequency Hopping Spread Spectrum Detection .....	1007
15.5 Spectraltool: Command Line Utility .....	1008
15.6 Spectral and Classifier Demo Application: athssd .....	1010
15.6.1 Athssd: User Arguments .....	1011
15.7 Spectral debug enhancements .....	1012
<b>16 Band Steering Methods .....</b>	1014
16.1 Transmit beamforming (TxBF) .....	1014
16.1.1 Explicit feedback TxBF .....	1015
16.1.2 Hardware definitions for beamforming .....	1019
16.1.3 Beamforming software flow .....	1028
16.1.4 Enable TxBF .....	1031
16.1.5 Change CV update period .....	1032
16.2 Single AP band steering .....	1032
16.2.1 Configuration basics .....	1034
16.2.2 Types of steering and conditions .....	1036
16.2.3 BTM compliance overview .....	1044

16.2.4	Steering safety and hysteresis . . . . .	1045
16.2.5	Advanced configuration parameters . . . . .	1047
16.2.6	Logging and debug CLI . . . . .	1048
16.2.7	Band steering architecture . . . . .	1052
16.2.8	Estimation algorithms . . . . .	1059
16.2.9	Disable blacklist functionality in band steering . . . . .	1059
16.3	Multi-AP coordinated steering and adaptive path selection . . . . .	1060
16.3.1	Multi-AP coordinated steering . . . . .	1061
16.3.2	Adaptive path selection . . . . .	1067
16.4	Suspension of probes in 2.4 GHz band . . . . .	1074
16.5	SSID steering . . . . .	1075
16.6	Disable Wi-Fi SON blacklist functionality in band steering . . . . .	1079
16.7	Band steering in QWRAP mode . . . . .	1080
16.7.1	Verify single AP band steering daemon is running, stadb logs and bandmon logs . . . . .	1081
16.7.2	Verify preassociation steering 11k/v clients based on 2.4GHz overload detection . . . . .	1081
16.8	Configure band steering per SSID pair . . . . .	1082
16.9	AP steering for legacy clients . . . . .	1083
16.9.1	Design principles . . . . .	1084
16.9.2	Configuration . . . . .	1087
16.9.3	Enhanced AP steering of 802.11k unfriendly clients as legacy clients . . . . .	1088
16.9.4	AP steering for legacy clients on QCA9886 chipsets . . . . .	1089
16.10	Display the hyd and lbd versions . . . . .	1089
16.11	Soft-blocking for safe connections . . . . .	1090
16.11.1	Design guidelines . . . . .	1091
16.11.2	Commands . . . . .	1092
<b>17</b>	<b>Repeater Access Point Functions . . . . .</b>	<b>1094</b>
17.1	PAPRD . . . . .	1094
17.1.1	Theory of Operation . . . . .	1094
17.1.2	Implementation . . . . .	1095
17.2	WDS . . . . .	1099
17.2.1	Theory of Operation . . . . .	1100
17.2.2	WDS Configuration Commands . . . . .	1101
17.2.3	Direct Attach WDS Implementation . . . . .	1101
17.2.4	802.11ac Offload WDS Implementation . . . . .	1102
17.2.5	NAWDS . . . . .	1102
17.3	Extender AP . . . . .	1109
17.3.1	Configuration . . . . .	1109
17.3.2	Feature Description . . . . .	1109
17.3.3	AP Configuration Mode . . . . .	1111
17.3.4	Top Level Design Overview . . . . .	1112
17.3.5	Frame processing . . . . .	1113

17.3.6 Implementation .....	1114
17.4 Proxy STA .....	1115
17.4.1 ProxySTA vs. WDS .....	1115
17.4.2 Terminology .....	1115
17.4.3 Theory of Design .....	1115
17.4.4 MAC Address Translation .....	1115
17.4.5 Configuration Tools Support for ProxySTA .....	1121
17.4.6 Configure ProxySTA .....	1121
17.4.7 Q-WRAP (Qualcomm Wireless Repeater AP) .....	1121
17.5 QWRAP support for Tri-Radio Standalone Repeater .....	1138
17.5.1 Feature requirement .....	1138
17.5.2 Design overview .....	1138
17.5.3 Data structure .....	1140
17.5.4 Route traffic always through primary radio .....	1141
17.5.5 Failsafe .....	1142
17.5.6 Wired client addition .....	1144
17.6 DBDC Repeater .....	1145
17.6.1 Challenges for DBDC Repeater .....	1145
17.6.2 Design for DBDC Repeater .....	1147
17.7 Special Packet handling for Loop avoidance and Load balancing .....	1148
17.7.1 Implementation of Tx and Rx data flows for DBDC repeater .....	1152
17.7.2 Configuration of Tx and Rx data flows for DBDC repeater .....	1152
17.8 Interface Manager (iface_mngr) application .....	1154
17.8.1 global_wds mode .....	1154
17.8.2 fast_lane mode .....	1156
17.9 Configure WPS for range extenders using a single push button .....	1157
17.10 Configure dynamic BSSID on repeater APs using wpa_cli .....	1158
17.10.1 Set BSSID before a STA VAP connection .....	1158
17.10.2 Set BSSID in the configuration file .....	1158
17.11 Automatic addition of Ethernet client to ProxySTA in QWRAP mode .....	1159
17.11.1 Configuration .....	1161
17.11.2 UCI configuration .....	1162
17.11.3 Limitations .....	1162
17.12 Enhanced Independent Repeater scan algorithm .....	1162
17.13 Support for same SSID on root and repeater APs in star topology in QWRAP .....	1164
17.13.1 Design overview .....	1164
17.13.2 Supported deployments .....	1165
17.13.3 Configuration .....	1166
17.13.4 Guidelines .....	1166
<b>18 Voice and Multimedia Features .....</b>	<b>1168</b>
18.1 WMM Flow Control and Buffer Management .....	1168
18.1.1 WMM Terminology .....	1168

18.1.2	Theory of Operation .....	1168
18.1.3	Implementation .....	1170
18.1.4	WMM & Receive Path .....	1174
18.1.5	Configuration .....	1175
18.2	Voice Enterprise .....	1175
18.2.1	Implementation .....	1176
18.2.2	802.11k .....	1177
18.2.3	Source Code .....	1180
18.3	Video over Wireless .....	1181
18.3.1	Acronyms .....	1181
18.3.2	VoW Features .....	1182
18.3.3	Video over Wireless (Offload) .....	1206
18.4	Smart Antenna .....	1213
18.4.1	Smart Antenna System .....	1213
18.4.2	Theory of Operation .....	1215
18.4.3	Implementation .....	1220
18.5	Unified Smart Antenna API .....	1227
18.5.1	Glossary .....	1227
18.5.2	Objectives .....	1227
18.5.3	Smart Antenna Interface Module .....	1227
18.5.4	Application Program Interface Functions .....	1227
18.5.5	Shared Data Structures .....	1236
18.5.6	Parameter Decoding .....	1238
<b>19</b>	<b>Wi-Fi SON Features .....</b>	<b>1246</b>
19.1	Range Extender Placement and Auto-configuration .....	1246
19.1.1	Range Extender Automatic Configuration and Mode Switching .....	1246
19.1.2	Range Extender Placement Assistance .....	1255
19.1.3	Using repacd .....	1261
19.1.4	Viewing Logs .....	1262
19.1.5	Sample scenario on LEDs placement .....	1262
19.2	Wi-Fi SON: Daisy chain and Best Uplink selection .....	1263
19.2.1	Information Elements .....	1264
19.3	Wi-Fi SON: Multi-SSID and Traffic Separation .....	1271
19.3.1	High Level Design .....	1271
19.3.2	Setup .....	1271
19.3.3	Configuration changes .....	1272
19.3.4	Driver changes .....	1273
19.4	Wi-Fi SON: Full Daisy Chain support across multiple hops .....	1273
19.5	Wi-Fi SON: Support across multiple Hy-Fi bridges .....	1280
19.6	Save STA information to flash for Wi-Fi SON .....	1283
19.7	Detection of WAN and LAN sample scenarios .....	1284
19.8	Network loop prevention sample scenarios .....	1288

19.9	Wi-Fi SON: Temporarily disable access interface . . . . .	1293
19.10	Enhanced WPS for RE to establish PBC connection in Wi-Fi SON . . . . .	1294
19.11	Fronthaul changes do not restart backhaul in Wi-Fi SON . . . . .	1297
19.12	Wi-Fi SON: Channel planning on 2.4 and 5 GHz bands for distributed Wi-Fi systems . . . . .	1298
19.12.1	Channel planning on 2.4 GHz bands . . . . .	1299
19.12.2	Channel planning on 5 GHz bands . . . . .	1300
19.12.3	Sample configuration scenarios . . . . .	1300
19.13	Wi-Fi SON: Ethernet roaming per port . . . . .	1302
19.14	Wi-Fi SON: PLC backhaul . . . . .	1302
19.14.1	PLC configuration . . . . .	1307
19.14.2	PLC SON interface mapping . . . . .	1308
19.14.3	Fronthaul AP VAPs and CAP reachability . . . . .	1308
19.14.4	Guidelines for configuring Wi-Fi SON interoperation with PLC . . . . .	1309
19.15	Wi-Fi SON with PLC: Support for daisy-chain and star topologies . . . . .	1309
19.15.1	Topology building process . . . . .	1311
19.15.2	Supported configuration scenarios . . . . .	1311
19.15.3	PLC Link Metrics . . . . .	1312
19.15.4	Band/Interface selection for AP steering . . . . .	1313
19.15.5	Configuration commands to enable Dynamic Link Metrics . . . . .	1313
19.15.6	Guidelines and limitations . . . . .	1314
19.16	Wi-Fi SON: Determine the join and leave events of peers . . . . .	1315
19.16.1	Addition of events to Wi-Fi SON module . . . . .	1315
19.16.2	Addition of sonevent module into HYD . . . . .	1315
19.16.3	Process of HYD communication with customer daemon . . . . .	1316
19.16.4	Use the customer library in the HYD customer module . . . . .	1316
19.16.5	Verify the join and leave events of one RE . . . . .	1317
19.16.6	Verify the join and leave events of two REs . . . . .	1318
19.17	Wi-Fi SON: Multinode WPS support for push button events . . . . .	1319
19.17.1	Supported configuration scenarios . . . . .	1319
19.17.2	Changes in behavior . . . . .	1319
19.17.3	Guidelines for working with multinode WPS . . . . .	1320
19.17.4	WPS Wi-Fi SON processing of the PBC press-event . . . . .	1320
19.17.5	Conditions when the WPS process is stopped . . . . .	1321
19.18	Wi-Fi SON: Cloning of backhaul VAP credentials for multi-SSID with traffic separation . . . . .	1321
19.19	Wi-Fi SON API for lbd blacklist . . . . .	1323
19.19.1	Sample configuration scenarios . . . . .	1326
19.20	Separate Wi-Fi SON AP steering threshold values for 2.4 GHz and 5 GHz bands . . . . .	1333
19.21	Best uplink algorithm for PLC interface metrics in a daisy chain Wi-Fi SON and PLC network . . . . .	1335
19.22	Interoperation of different PLC chipset models in a Wi-Fi SON network . . . . .	1342
19.23	Wi-Fi SON: Skip PFS and improve airtime calculation . . . . .	1345

19.23.1	Overview of steering features . . . . .	1346
19.23.2	Design and implementation . . . . .	1347
19.23.3	Verify the steering enhancements . . . . .	1348
19.24	Support for REH172 as a root AP . . . . .	1349
19.24.1	REH172 overview . . . . .	1349
19.24.2	Software architecture overview . . . . .	1351
19.24.3	REH172 in routing mode . . . . .	1352
19.25	Wi-Fi SON: Diagnostics and troubleshooting . . . . .	1357
19.25.1	Wi-Fi SON diagnostics design . . . . .	1357
19.25.2	Wi-Fi SON diagnostics algorithm . . . . .	1357
19.25.3	Functionality and components . . . . .	1358
19.25.4	Configure Wi-Fi SON diagnostics . . . . .	1359
<b>20</b>	<b>Coexistence of WLAN with NoDS mesh, BT, ZigBee, and CSRmesh . . . . .</b>	<b>1360</b>
20.1	NoDS Frame support for Mesh feature . . . . .	1360
20.1.1	Requirements . . . . .	1360
20.1.2	High Level Design . . . . .	1361
20.1.3	Additional feature list . . . . .	1365
20.2	Bluetooth WLAN coexistence . . . . .	1370
20.2.1	Assumptions . . . . .	1370
20.2.2	COEX initialization . . . . .	1371
20.2.3	Priority table . . . . .	1372
20.2.4	COEX module . . . . .	1372
20.2.5	Beacon frame handling during COEX . . . . .	1373
20.2.6	Management Frame Handling during COEX . . . . .	1374
20.2.7	WMM_AC_VO Frame Handling during COEX . . . . .	1375
20.2.8	HOST Driver Interface . . . . .	1375
20.3	Coexistence of WLAN and Bluetooth for CSRMesh . . . . .	1376
20.4	Coexistence of WLAN and ZigBee for CSRMesh . . . . .	1399
20.5	Coexistence of Bluetooth and ZigBee with WLAN on IPQ8064 . . . . .	1419
20.5.1	Behavior of hardware interfaces for Bluetooth/Zigbee and WLAN coexistence .	1419
20.5.2	TDM operations for coexistence . . . . .	1420
20.5.3	Baseline priority table for COEX . . . . .	1422
20.5.4	Configure BT coexistence . . . . .	1422
20.5.5	Configure BT on the two devices . . . . .	1423
20.5.6	Load firmware image . . . . .	1423
20.6	LTE gateway and LTE coexistence . . . . .	1424
20.6.1	LTE gateway configuration GUI . . . . .	1426
<b>21</b>	<b>WiGig (802.11ad Standard) Features . . . . .</b>	<b>1428</b>
21.1	802.11ad WMI commands for management packets and antenna control . . . . .	1428
21.1.1	Management packets retry limit . . . . .	1428
21.1.2	Antenna control . . . . .	1429

21.1.3	Antenna state readback for 802.11ad . . . . .	1432
21.1.4	WMI command IDs . . . . .	1434
21.1.5	WMI event IDs . . . . .	1435
21.2	802.11ad firmware recovery . . . . .	1435
21.2.1	Control commands . . . . .	1436
21.3	802.11ad: Fast session transfer . . . . .	1436
21.3.1	Configure the AP160 for 802.11ad . . . . .	1437
21.3.2	Connect an 802.11ad STA to an 802.11ad AP . . . . .	1438
21.3.3	Enter RF-kill state . . . . .	1439
21.3.4	Enter automatic channel selection (ACS) . . . . .	1439
21.3.5	Enable Wi-Fi protected setup (WPS) functionality . . . . .	1439
21.3.6	Enable fast session transfer (FST) functionality . . . . .	1440
21.3.7	802.11ad: Board file configuration . . . . .	1441
21.4	Massive array for 802.11ad . . . . .	1442
21.5	802.11ad system overview . . . . .	1443
21.5.1	Wi-Fi system . . . . .	1445
21.5.2	Qualcomm 60 GHz device architecture . . . . .	1446
21.5.3	Software partition . . . . .	1447
21.6	Fast session transfer . . . . .	1451
21.6.1	FST architecture . . . . .	1452
21.6.2	FST flows . . . . .	1454
21.6.3	FST network topology . . . . .	1457
21.6.4	FST manager . . . . .	1457
21.6.5	Debug logs . . . . .	1457
<b>22</b>	<b>Logging and Statistics for Debugging . . . . .</b>	<b>1460</b>
22.1	Packet Logging . . . . .	1460
22.1.1	Packet Logging for Direct Attach . . . . .	1460
22.1.2	Packet Logging for Offloaded Architecture . . . . .	1462
22.2	AP Statistics . . . . .	1464
22.2.1	AP Statistics Implementation . . . . .	1465
22.2.2	AP Statistics Usage . . . . .	1466
22.2.3	AP Statistics Listing . . . . .	1469
22.2.4	Important Notes . . . . .	1472
22.2.5	Sample apstats Output . . . . .	1473
22.2.6	Athstats Enhancement for 802.11ac . . . . .	1477
22.3	Tracking latency and loss of 802.11 packets . . . . .	1479
22.4	Enhanced station statistics display for Tx and Rx frames . . . . .	1480
22.5	Customer WLAN event reports . . . . .	1482
22.6	AP Diagnostics for Carrier . . . . .	1485
22.6.1	Configuration . . . . .	1486
22.6.2	High-level design . . . . .	1486
22.7	Generate target core dump . . . . .	1488

22.8	Enhanced client statistics display for RSSI, Tx/Rx packets, and STBC . . . . .	1489
22.8.1	Data structure changes . . . . .	1489
22.8.2	Sample output of apstats command . . . . .	1492
22.9	Connection state logging functionality . . . . .	1494
22.10	Smart logging method . . . . .	1495
22.10.1	Host-FW interaction . . . . .	1495
22.10.2	Identification of trigger conditions . . . . .	1497
22.11	CE logging functionality . . . . .	1498
22.11.1	Storage space of logs . . . . .	1499
22.11.2	Storage format of logs . . . . .	1499
22.11.3	Retrieval of logs . . . . .	1500
22.11.4	Usage information . . . . .	1500
22.11.5	Examples of CE logs . . . . .	1500
22.11.6	Examples of smart logs . . . . .	1501
22.12	Buffer tracking . . . . .	1503
22.13	Packet log and Tx completion message correlation to mirror Tx data packets on QCA9880 1503	
22.13.1	Unique ID for Tx Completion Indication . . . . .	1505
22.13.2	Tx Completion Indication . . . . .	1505
22.13.3	Management Tx Completion Indication . . . . .	1506
22.13.4	Packet log message . . . . .	1507
22.13.5	Call flows . . . . .	1509
22.14	Enhanced client statistics display for Tx management frames, minimum/maximum RSSI, and received data MPDUs 1511	
22.15	Integration of Wi-Fi statistics with vendors for extended statistics . . . . .	1512
22.16	Display of decoded watchdog timer signature for QCA9980 . . . . .	1530
22.16.1	Mapping of decoded watchdog timer debug log . . . . .	1532
22.17	Smart logging for HALPHY modules . . . . .	1537
22.17.1	Host-FW interaction . . . . .	1537
22.17.2	Implementation details . . . . .	1538
22.17.3	Identification of calibration triggers—firmware . . . . .	1538
22.17.4	Identification of non-calibration triggers—Firmware . . . . .	1541
22.18	WMI support for debug statements in UTF . . . . .	1543
22.19	Display latency for each PPDU using packet log tool . . . . .	1543
22.19.1	Latency information . . . . .	1544
22.19.2	Pktlog –R . . . . .	1544
22.19.3	Sample test scenario . . . . .	1545
22.20	Enhanced TID queue statistics . . . . .	1548
22.21	Enhanced client statistics for acknowledgment failures, modulation rates, authentication details 1551	
22.21.1	Station connection parameters . . . . .	1559

## List of Tables

Table 3-1 Synchronizing Methods .....	132
Table 3-2 WBUF Types .....	133
Table 4-1 Build Options .....	180
Table 4-2 VAP commands .....	187
Table 4-3 List VAP Command .....	188
Table 4-4 Delete VAP Command .....	188
Table 4-5 Iwconfig Commands .....	189
Table 5-1 APONLY Switch Points .....	213
Table 6-1 Network configuration .....	342
Table 6-2 Cumulative crypto statistics .....	348
Table 6-3 Engine level crypto statistics .....	348
Table 6-4 Session Level crypto statistics .....	348
Table 6-5 Crypto configuration information .....	349
Table 6-6 Crypto Control information .....	349
Table 6-7 Rx MAC Flow Control .....	415
Table 6-8 Mapping between load balance value and CPU core .....	442
Table 6-9 UDF profile configuration format .....	446
Table 6-10 ACL rule confirmation format .....	446
Table 6-11 ARP configuration format .....	450
Table 6-12 Interface entry configuration format .....	452
Table 8-1 Tx power-out in proposed scenario .....	533
Table 8-2 Target Power less than CTL Power .....	534
Table 8-3 Target Power equal to CTL Power .....	535
Table 8-4 Target Power greater than CTL Power .....	535
Table 8-5 Physical layer parameters .....	536
Table 10-1 Strict queue and fair queue ATF algorithm .....	618
Table 11-1 802.11ac features .....	696
Table 11-2 Maximum NSS based on chain-mask and bandwidth for QCA9984 .....	709
Table 11-3 Maximum NSS based on chain-mask and bandwidth for QCA9886 .....	709
Table 11-4 QCA9984 table .....	710
Table 11-5 QCA9886 Table .....	710
Table 11-6 NSS/BW negotiated with non-QCA9984 solutions .....	712
Table 11-7 NSS/BW negotiated with non-QCA9886 solutions .....	712
Table 11-8 Protected management frames (PMF) parameters .....	726
Table 13-1 OCE IE Format .....	776
Table 13-2 OCE Attribute General Format .....	776
Table 13-3 OCE Attributes .....	777
Table 13-4 OCE Capability Indication Attribute .....	777
Table 13-5 OCE Control field as advertised by an OCE AP .....	777
Table 13-6 OCE Control field as advertised by an OCE STA .....	778
Table 13-7 RSSI-based Association Rejection Attribute .....	779

Table 13-8 Reduced WAN Metrics Attribute .....	779
Table 13-9 Terms and definitions for Wi-Fi multiband operation .....	789
Table 13-10 MBO-OCE IE Format .....	801
Table 13-11 MBO Attribute General Format .....	802
Table 13-12 MBO Attributes .....	802
Table 13-13 MBO AP Capability Indication Attribute .....	803
Table 13-14 MBO AP Capability Indication Field Values .....	803
Table 13-15 Non-preferred Channel Report Attribute .....	803
Table 13-16 Preference Field Values .....	804
Table 13-17 Reason Code Field Values .....	804
Table 13-18 MBO Cellular Capabilities Attribute .....	805
Table 13-19 Cellular Data Connectivity Field .....	805
Table 13-20 Association Disallowed Attribute .....	806
Table 13-21 Reason Code Field Values .....	806
Table 13-22 Cellular Data Connection Preference Attribute .....	806
Table 13-23 Cellular Preference Field Values .....	806
Table 13-24 Transition Reason Code Attribute .....	807
Table 13-25 Transition Reason Code Field Values .....	807
Table 13-26 Transition Rejection Reason Code Attribute .....	808
Table 13-27 Transition Rejection Reason Code Field Values .....	808
Table 13-28 Association Retry Delay Attribute .....	809
Table 13-29 MBO ANQP-elements .....	809
Table 13-30 MBO ANQP-element Subtype Values .....	810
Table 13-31 Non-preferred Channel Report Subelement .....	811
Table 13-32 Table 4-23: Cellular Data Capabilities Subelement .....	812
Table 13-33 athtestcmd Frequently Used Options .....	827
Table 13-34 athtestcmd Tx Rate Table .....	831
Table 13-35 Packet type and subtype configuration .....	843
Table 14-1 Radartool Command Line Options .....	901
Table 14-2 DFS Debug Levels .....	902
Table 14-3 iwpriv CLI Commands .....	920
Table 14-4 Configure RSSI scan of NACs parameters .....	967
Table 15-1 Spectral scan parameters .....	979
Table 15-2 Static 20 Mode .....	980
Table 15-3 Dynamic 20/40 Mode .....	981
Table 15-4 FFT Size vs. Bandwidth Modes .....	982
Table 15-5 Search FFT Report .....	987
Table 15-6 Search FFT report (QCA9984/QCA9888 160/80+80 MHz chipset) .....	987
Table 15-7 Spectral Scan Summary Report (QCA999x/998x) .....	988
Table 15-8 Spectral Scan Summary Report (QCA9984/QCA9888 160/80+80 MHz chipset) .....	989
Table 15-9 Generation II Spectral Scan Configuration Register .....	990
Table 15-10 Spectral Indirection .....	993
Table 15-11 Spectral Indirection Table Mapping .....	994

Table 15-12 Spectral IOCTLs .....	995
Table 15-13 Spectraltool Commands .....	1008
Table 15-14 Athssd User Arguments .....	1011
Table 16-1 Feature comparison between phases .....	1033
Table 16-2 Top-level configuration .....	1035
Table 16-3 Pre-Association Steering Parameters .....	1036
Table 16-4 Basic Idle non-overload steering parameters .....	1038
Table 16-5 Basic active steering configuration parameters .....	1040
Table 16-6 Basic Interference Avoidance Steering configuration parameters .....	1043
Table 16-7 BTM compliance configuration parameters .....	1044
Table 16-8 Configuration parameters .....	1046
Table 16-9 Advanced configuration parameters .....	1047
Table 16-10 Available debug logging areas .....	1049
Table 16-11 Softblocking parameters .....	1093
Table 16-12 Softblocking parameters .....	1093
Table 17-1 Maximum number of wired and wireless clients .....	1123
Table 17-2 TX data flow of DBDC Repeater based on different configurations .....	1151
Table 17-3 RX data flow of DBDC Repeater based on different configurations .....	1151
Table 18-1 802.1D Priority to AC Mappings .....	1169
Table 18-2 TID to AC Conversion .....	1169
Table 18-3 Smart Antenna Parameter Configuration Commands .....	1224
Table 18-4 Tx Feedback Decoding for 5 GHz Radio .....	1240
Table 19-1 repacd configuration parameters relevant for auto-configuration .....	1253
Table 19-2 wsplcd configuration parameters relevant for auto-configuration .....	1254
Table 19-3 Example LED schemes for RE devices .....	1257
Table 19-4 Example LED schemes for client devices .....	1258
Table 19-5 Parameters for LED indications .....	1259
Table 19-6 Channel Info Request MME .....	1342
Table 19-7 Channel Info Confirm MME .....	1343
Table 19-8 CHANNEL_DATA_RATES .....	1343
Table 19-9 CHANNEL_MEDIUM_UTILIZATION .....	1344
Table 20-1 Priority table .....	1372
Table 20-1 Priority table for BT and WLAN .....	1422
Table 21-1 Acronyms .....	1435
Table 22-1 Type and subtypes .....	1497
Table 22-2 Offload statistics parameters .....	1513
Table 22-3 Failure type and subtypes .....	1537
Table 22-4 Station connection parameters .....	1559

## List of Figures

Figure 1-1 Direct Attach Architecture .....	55
Figure 1-2 Full Offload Architecture .....	56
Figure 1-3 WLAN Software Layers .....	57
Figure 1-4 Offload stack diagram .....	61
Figure 1-5 Host-to-Target interface diagram .....	62
Figure 1-6 Driver - Block Diagram until QCA_Networking_2016.SPF.2.0 release .....	63
Figure 1-7 Driver - Block Diagram until QCA_Networking_2016.SPF.3.0 release .....	65
Figure 1-8 Block Diagram in QCA_Networking_2016.SPF.4.0 release and later .....	66
Figure 1-9 Block Diagram Comparison .....	67
Figure 1-10 Driver - Block Diagram in QCA_Networking_2016.SPF.3.0 .....	67
Figure 1-11 Phase 1 Changes .....	68
Figure 1-12 WLAN Driver Sizes .....	76
Figure 2-1 WLAN Driver Major Objects .....	78
Figure 2-2 WIN WMI design .....	92
Figure 2-3 WMI interactions .....	93
Figure 2-4 WMI Transmit showing vdev create using WMI command WMI_VDEV_CREATE_CMDID 94	
Figure 2-5 WMI rx path for WIN design .....	95
Figure 3-1 Descriptor List .....	134
Figure 3-2 Transmit Queue Mapping .....	137
Figure 3-3 Radio Interface Initialization (PCI Mode) .....	139
Figure 3-4 VAP Create Codeflow .....	141
Figure 3-5 VAP Delete Codeflow .....	142
Figure 3-6 VAP Start Codeflow .....	143
Figure 3-7 VAP Stop Codeflow .....	144
Figure 3-8 Data Transmit Flow .....	148
Figure 3-9 Data Transmit Completion Codeflow .....	149
Figure 3-10 Receive Codeflow (Chart 1) .....	150
Figure 3-11 Receive Codeflow (Chart 2) .....	151
Figure 3-12 TxRx Architectural Overview .....	155
Figure 3-13 Transmit Path Flow .....	157
Figure 3-14 Transmit Completion Flow .....	158
Figure 3-15 Rx Processing Flow (Part 1) .....	159
Figure 3-16 Rx Processing Flow (Part 2) .....	160
Figure 3-17 Offload Layer Positioning .....	163
Figure 3-18 Flow steering in IPQ4019 ESS .....	168
Figure 3-19 RPS/RFS in Linux kernel .....	169
Figure 4-1 Configuration API Support Architecture .....	191
Figure 5-1 AP Mode Operation .....	206
Figure 5-2 APONLY Data Path .....	211
Figure 5-3 Client Mode Operation .....	215

Figure 5-4 Driver Interface for Hostap . . . . .	219
Figure 5-5 Multiple BSSID Addresses . . . . .	229
Figure 5-6 VLAN Block Diagram . . . . .	233
Figure 5-7 ieee80211_classify Function . . . . .	233
Figure 5-8 osif_deliver_data Function . . . . .	234
Figure 5-9 Hy-Fi Software Architecture . . . . .	236
Figure 5-10 LOWI overview . . . . .	240
Figure 5-11 LOWI client side library architecture . . . . .	241
Figure 5-12 LOWI server architecture . . . . .	243
Figure 5-13 Block Diagram of LOWI on Access Point . . . . .	292
Figure 5-14 Single Sided RTT example . . . . .	293
Figure 5-15 Double Sided RTT example . . . . .	294
Figure 5-16 Example call flow in HOST driver for Measurement request/response . . . . .	297
Figure 6-1 Offload Optimized Transmit Flow Diagram . . . . .	312
Figure 6-2 Transmit Completion and Scheduling Flow Diagram . . . . .	313
Figure 6-3 HNAT Block Diagram . . . . .	317
Figure 6-4 Load distribution among components . . . . .	324
Figure 6-5 Campus Deployment using IPsec . . . . .	329
Figure 6-6 Remote branch office deployment . . . . .	330
Figure 6-7 Crypto offload in context of IPsec . . . . .	331
Figure 6-8 IPsec acceleration model for IPQ806x . . . . .	332
Figure 6-9 IPsec offload configuration of NSS firmware . . . . .	333
Figure 6-10 Connection topology for a “2” tunnel IPsec setup . . . . .	336
Figure 6-11 Connection topology for a “1” tunnel IPsec setup . . . . .	337
Figure 6-12 Subnets and IP(s) used in the “2” tunnel IPsec topology . . . . .	337
Figure 6-13 Subnets and IP(s) used in the “1” tunnel IPsec topology . . . . .	338
Figure 6-14 Traffic pattern and the corresponding MTU(s) . . . . .	338
Figure 6-15 Connection topology for a “2” tunnel IPsec + WLAN setup . . . . .	342
Figure 6-16 Host-target connectivity over CAT5 cable . . . . .	357
Figure 6-17 MII boot . . . . .	358
Figure 6-18 Native Dual-Stack Co-existence Mechanism . . . . .	378
Figure 6-19 IPv6 Rapid Deployment (6RD) Mechanism . . . . .	378
Figure 6-20 Dual Stack Lite (DS-Lite) Mechanism - IPv4 Over IPv6 . . . . .	379
Figure 6-21 Native dual stack co-existence mechanism . . . . .	388
Figure 6-22 6RD coexistence mechanism . . . . .	389
Figure 6-23 Dual stack lite coexistence mechanism . . . . .	389
Figure 6-24 Sample Topology . . . . .	419
Figure 6-25 ACL window rule . . . . .	442
Figure 6-26 IPv4 GRE tunnel packet . . . . .	444
Figure 7-1 rc_Find_ht() Algorithm . . . . .	472
Figure 7-2 RCUpdate() Algorithm . . . . .	475
Figure 7-3 RATE_GetTxRetrySched Algorithm . . . . .	479
Figure 8-1 UAPSD Mechanism . . . . .	493

Figure 8-2 UAPSD Processing .....	495
Figure 8-3 WLAN Firmware .....	526
Figure 9-1 Failure Case CAT and COT .....	580
Figure 10-1 Calibration Setup .....	600
Figure 10-2 ART2 Architecture .....	601
Figure 10-3 ART2 Client Setup .....	601
Figure 10-4 ART2 AP Setup .....	602
Figure 10-5 SIFS Burst Overview .....	606
Figure 10-6 Sequence Flow .....	614
Figure 10-7 ATF Implementation Part 1 .....	616
Figure 10-8 ATF Implementation Part 2 .....	617
Figure 10-9 Firmware signature validation high level flowchart .....	655
Figure 10-10 High-level peer-based slow schedule design .....	683
Figure 10-11 SW/FW architecture with global flow control .....	685
Figure 10-12 SW/FW architecture with lookahead feature .....	686
Figure 10-13 Tx Post/completion timing diagram .....	691
Figure 10-14 Function in the Latency sensitive path .....	692
Figure 11-1 Frequencies allowed for 80+80 bandwidth when RegDomain Extension Register Bit = 1	701
Figure 11-2 Frequencies allowed for 160 MHz bandwidth .....	702
Figure 11-3 PMF Block Diagram .....	719
Figure 12-1 Example depicting conversion for four clients .....	734
Figure 13-1 Overview of Software Retry .....	753
Figure 13-2 Example of Hardware Filtering .....	753
Figure 13-3 Implementation of Hardware Filtering Logic .....	755
Figure 13-4 Flow of ath_tx_mpdu_requeue .....	756
Figure 13-5 Handling PS-Poll (UMAC Part) .....	757
Figure 13-6 PS-Poll Handling (LMAC Part) .....	757
Figure 13-7 Staggered Beaoning Scheme .....	758
Figure 13-8 Bursted Beaoning Scheme .....	759
Figure 13-9 WoW Packet Format .....	761
Figure 13-10 Typical OCE Network Topology .....	770
Figure 13-11 Available Capacity field in Reduced WAN Metrics Attribute .....	779
Figure 13-12 Flow of updating SSID info for beacon received with hidden SSID .....	784
Figure 13-13 Work flow of neighbor report table configuration .....	787
Figure 13-14 Work flow of inclusion of IEs to BTM request frames .....	788
Figure 13-15 MBO Wireless Infrastructure Network .....	791
Figure 13-16 MBO Wireless Infrastructure Network with Cellular Access Network .....	792
Figure 13-17 MBO Query List ANQP-element Payload Format .....	810
Figure 13-18 WNM-Notification Request frame format .....	811
Figure 13-19 Flow of Suspend and Resume beacon .....	816
Figure 13-20 Workflow of suspension and resumption of beacons .....	819
Figure 13-21 TX99 11n Mode Overview .....	821
Figure 13-22 TX99 11ac Mode Overview .....	826

Figure 13-23 Access Points Controlled by Access Controller .....	838
Figure 13-24 Tx Raw frame .....	843
Figure 13-25 Tx side Mapping to MSDU Link Extension Descriptor .....	845
Figure 13-26 Three-certificate mode .....	863
Figure 14-1 DFS Overview .....	881
Figure 14-2 DFS Operation .....	883
Figure 14-3 CSA and Local Radar handling in Repeater AP .....	886
Figure 14-4 Modified Assoc state machine to support Repeater DFS .....	887
Figure 14-5 Enhanced DFS support for RE to inform Root AP of radar detection .....	889
Figure 14-6 Zero CAC DFS Events .....	891
Figure 14-7 Vap down .....	891
Figure 14-8 Channel change .....	892
Figure 14-9 PreCAC NOL timeout .....	893
Figure 14-10 PreCAC Timeout .....	893
Figure 14-11 Radar detection on first and second Segment .....	894
Figure 14-12 DFS Attach .....	897
Figure 14-13 DFS Detach .....	898
Figure 14-14 ioctl Control .....	899
Figure 14-15 DFS Radar Detection Flow .....	900
Figure 14-16 802.11NG EACS PLUS Algorithm .....	917
Figure 14-17 802.11NA/AC EACS Plus Algorithm (Part 1) .....	918
Figure 14-18 EACS Plus Algorithm (Part 2) .....	919
Figure 14-19 ICM block diagram .....	922
Figure 14-20 ICM State Machine .....	923
Figure 14-21 Channel loading functionality .....	935
Figure 14-22 Channel Hopping Algorithm .....	936
Figure 14-23 Scenario for cloud based trigger .....	959
Figure 14-24 Flow diagram describing AP and STA VAP behavior during topology change for Scan mode .....	961
Figure 15-1 Illustration of Spectral Scan with <i>spectral_scan_short_rpt</i> Disabled .....	978
Figure 15-2 Illustration of Spectral Scan with <i>spectral_scan_short_rpt</i> Enabled .....	979
Figure 15-3 Spectral Scan FFT computed at Full ADC Rate .....	982
Figure 15-4 Determination of Strong Bins for Wideband/Narrowband Classification during Spectral FFT .....	983
Figure 15-5 Overview of Spectral TLV Formats .....	986
Figure 15-6 Host Spectral Driver Architecture .....	993
Figure 15-7 WMI Configuration and Event Sequencing for Spectral .....	999
Figure 15-8 MWO Interference Pattern In Channel 11 .....	1002
Figure 15-9 RSSI Pattern of MWO Interference .....	1002
Figure 15-10 Video Bridge Interference in Channel 6 .....	1004
Figure 15-11 WLAN Interference in Channel 6 .....	1005
Figure 15-12 DSSS Cordless Phone Interference in Channel 161 .....	1006
Figure 15-13 FHSS Cordless Phone Interference in Channel 6 .....	1007

Figure 15-14 athssd Spectral Analysis Setup .....	1011
Figure 15-15 State transition diagram for athssd .....	1012
Figure 16-1 Explicit Feedback TxBF .....	1016
Figure 16-2 Interference Detection Pipeline .....	1042
Figure 16-3 Load balancing daemon high-level architecture .....	1053
Figure 16-4 Implementation - Step 1 .....	1077
Figure 16-5 implementation - Step 2 .....	1078
Figure 16-6 Implementation - Step 3 .....	1079
Figure 17-1 PA Distortion .....	1094
Figure 17-2 PAPRD Algorithm Flowchart .....	1095
Figure 17-3 RvR Throughput with and without PAPRD .....	1097
Figure 17-4 WDS Block Diagram .....	1100
Figure 17-5 Typical NAWDS Scenario .....	1102
Figure 17-6 Typical WDS Block Diagram .....	1103
Figure 17-7 NAWDS Connections .....	1104
Figure 17-8 Extender AP Block Diagram .....	1109
Figure 17-9 Extender AP Use Case .....	1110
Figure 17-10 Extender AP Configuration Mode .....	1111
Figure 17-11 AP Extender Overview .....	1112
Figure 17-12 ARP Packet Format .....	1118
Figure 17-13 Q-WRAP Overview .....	1122
Figure 17-14 Q-WRAP L2 bridging design .....	1125
Figure 17-15 Uplink – Non-Isolation .....	1126
Figure 17-16 Downlink – Non-isolation .....	1127
Figure 17-17 Isolation mode .....	1128
Figure 17-18 wrapd Block Diagram .....	1131
Figure 17-19 WRAPD master-slave model .....	1139
Figure 17-20 WRAPD data structure design .....	1141
Figure 17-21 Scenario - MPSTA disconnected .....	1143
Figure 17-22 Scenario - MPSTA connected .....	1144
Figure 18-1 QoS Control Field .....	1169
Figure 18-2 Packet Classification .....	1170
Figure 18-3 axq_minfree Initialization .....	1171
Figure 18-4 WMM Flow Control Queue Initialization .....	1171
Figure 18-5 WMM Flow Control During Transmission .....	1172
Figure 18-6 WMM Flow Control After Transmission Completion .....	1172
Figure 18-7 Tx flow with WMM Flow Control .....	1173
Figure 18-8 Tx Completion with WMM Flow Control Buffers Update .....	1174
Figure 18-9 ath_rx_timer Timeout Initialization .....	1174
Figure 18-10 Node Rx Initialization for WMM .....	1175
Figure 18-11 802.11k High Level Diagram .....	1178
Figure 18-12 Retry Reordering .....	1183
Figure 18-13 Ethernet Switch Example .....	1188

Figure 18-14 Flow Control Data Structures .....	1199
Figure 18-15 APH126 Reference Design Block Diagram .....	1201
Figure 18-16 WMM Configuration (HCM Framework) .....	1208
Figure 18-17 Penalization .....	1211
Figure 18-18 VoW Bin Thresholds .....	1212
Figure 18-19 Smart Antenna Block Diagram (3x3 Chipset Shown) .....	1214
Figure 18-20 Retraining Algorithm .....	1219
Figure 18-21 Performance-based Trigger Flowchart .....	1220
Figure 18-22 UMAC Training Initialization Sequence .....	1221
Figure 18-23 UMAC Traffic Generation (Insufficient Traffic Case) .....	1221
Figure 18-24 Antenna Selection when Training using Existing Traffic .....	1222
Figure 18-25 Training Packets Selection in LMAC Transmit Path .....	1223
Figure 18-26 Training Stats Collection in Transmit Done Path .....	1224
Figure 19-1 State machine for LED configuration process on RE devices .....	1256
Figure 19-2 State Machine for LED Configuration Process on Client Devices .....	1257
Figure 19-3 RE Movement from SON to DAISY SON Mode .....	1267
Figure 19-4 RE Movement from DAISY SON to SON Mode .....	1267
Figure 19-5 Hop2 RE 5 GHz has a Bad link .....	1268
Figure 19-6 1905 Topology message exchange example .....	1275
Figure 19-7 Daisy chain support for interconnection of CAP and RE2 using Wi-Fi and PLC, and RE1 using Wi-Fi only .....	1310
Figure 19-8 Daisy chain support for interconnection of CAP, RE1, and RE2 using PLC and Wi-Fi .....	1310
Figure 19-9 Star topology for interconnection of REs and CAP with Wi-Fi SON and PLC .....	1310
Figure 19-10 PLC daisy chain with RE1 as DK01/DK07 .....	1336
Figure 19-11 PLC daisy chain with RE1 as REH172 .....	1336
Figure 19-12 PLC daisy chain becomes a star topology after selecting PLC as backhaul in Daisy Chain topology .....	1336
Figure 19-13 PLC daisy chain becomes pure WLAN daisy chain topology for CAP-RE1-RE2 after selecting WLAN interface as backhaul .....	1337
Figure 19-14 PLC daisy chain becomes a star topology after selecting PLC as backhaul in Daisy Chain topology .....	1337
Figure 19-15 PLC daisy chain becomes pure WLAN daisy chain topology for CAP-RE1-RE2 after selecting WLAN interface as backhaul .....	1337
Figure 19-16 Software architecture .....	1351
Figure 19-17 IPQ40x8 hardware architecture .....	1351
Figure 19-18 DK05 Default Port association in each group .....	1353
Figure 21-1 Integrated HMC and RF module .....	1444
Figure 21-2 HMC with RF and JTAG connected to auxiliary PCIe slot .....	1444
Figure 21-3 Wi-Fi system examples .....	1445
Figure 21-4 Qualcomm 60 GHz device high-level architecture .....	1446
Figure 21-5 Software high-level architecture .....	1447
Figure 21-6 Windows software partition .....	1448
Figure 21-7 Firmware architecture .....	1450
Figure 21-8 μCode architecture .....	1451

Figure 21-9 FST Architecture . . . . .	1452
Figure 21-10 FST Control Path and Data Path . . . . .	1453
Figure 21-11 High-Level Flow when FST-Capable Clients Connect . . . . .	1454
Figure 21-12 FST Session Flow (Control Path) . . . . .	1456
Figure 21-13 FST network topology . . . . .	1457
Figure 22-1 Packet-Log PER/Rate Report . . . . .	1462
Figure 22-2 High Level Architecture Diagram. . . . .	1541

QUALCOMM®  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

# Preface

---

Qualcomm Technologies Access Point (AP) software is largely based on the Linux operating system, and consists of the following components, apart from the standard Linux operating system kernel:

- Wireless LAN
- Ethernet
- Router Stack
- Hybrid Network

This document is intended to help AP programmers understand, enhance, or modify AP software as required. This guide provides a high-level overview of the AP system architecture, and focuses on the design and implementation of the Wireless LAN component.

## About this Document

This document comprises the following chapters and appendixes:

<a href="#">Chapter 1, WLAN AP Driver Architecture</a>	Provides an overview of the major software components.
<a href="#">Chapter 2, WLAN AP Driver Layers</a>	Explains the WLAN AP driver layers and the corresponding data structures or objects for these layers. It also describes the code base structure and provides information on directory and common data structure, layers, and source tree layout.
<a href="#">Chapter 3, WLAN AP Driver Operations</a>	Describes the build options that are used with the makefile utility to construct and invoke the driver, and the runtime configuration methods that is required for proper device operations.
<a href="#">Chapter 4, WLAN AP Driver Build and Configuration Methods</a>	Describes the WLAN AP driver operations, such as the processing contexts, buffer abstraction, handling of transmission queues to support prioritization, and the code and data path flows.
<a href="#">Chapter 5, WLAN AP Modes</a>	Describes the different modes or variants of operation of WLAN APs, such as AP mode, client mode and security, multiple basic service set identifier (BSSID), virtual local area network (VLAN) support, Hy-Fi WLAN, AP-only mode, Positioning: LOWI on AP, Location wireless interface (LOWI).

<a href="#">Chapter 6, Acceleration and Offload Features</a>	Describes the offload functionalities, such as TCP Segmentation Offload, Large Receive offload, checksum offload, Offload Optimized Host Data Path, NSS offload, and HNAT Wi-Fi offload.
<a href="#">Chapter 7, Rate Control Features</a>	Describes the rate control functionalities and techniques, such as 256 QAM rate support in 2.4 GHz operation, coordinated ATF between RootAP & Repeater, disable selected legacy rates, and content aware routing.
<a href="#">Chapter 8, Power Management Techniques</a>	Describes the power management mechanisms, such as UAPSD WMM Power Save (WMM-PS), Transmit Power Control (TPC), Advanced Enterprise for WLAN driver version 10.4, and thermal mitigation.
<a href="#">Chapter 9, Regulatory Compliance of WLAN APs</a>	Describes the compliance of WLAN APs with the global regulatory agencies, such as European Telecommunications Standards Institute (ETSI), Federal Communications Commission (FCC), Japan Ministry of Internal Affairs and Communications (MIC) regulations, and also individual countries at all times.
<a href="#">Chapter 10, Memory and Performance</a>	Describes the memory and performance characteristics with WLAN AP operations, such as ART2 calibration, interrupt mitigation, SIFS Burst, Adaptive Noise Immunity (ANI), scatter/gather DMA, Airtime Fairness, Peer Flow Control Data Path, QCache, Firmware Code Sign, firmware authentication at host, WLAN LED Implementation, and preallocation of the required runtime memory.
<a href="#">Chapter 11, IEEE 802.11 Features</a>	Describes the different IEEE 802.11 specifications and protocols that are supported, such as 802.11ac, 802.11n, 802.11w Protected Management Frames (PMF), and 802.11ad (WiGig).
<a href="#">Chapter 12, QoS and Policy Control Operations</a>	Describes the quality of service (QoS) capabilities, such as access control lists (ACLs) and iQue (Intelligent QoS for User Experience), to specify policies and filters for controlling and classifying the manner in which traffic must be forwarded for processing.
<a href="#">Chapter 13, Beacons and Frames Transmission</a>	Discusses the functionalities supported in the transmission of beacons, probe request frames, association frames, and probe response frames, Quick STA Kickout (QSK), Software Retry (SWR), bursted beaconing, Dynamic Transmit Chainmask Selection (DTCS), HotSpot 2.0, TX99 Mode, Raw Mode Tx/Rx, Wake on Wireless AP Assist, Host Tx Flow Control, Descriptor configuration, and dynamic beacons.
<a href="#">Chapter 14, Channel Selection</a>	Offers an overview of the channel selection methodologies and techniques, such as Channel Width Management, Dynamic Frequency Selection (DFS), EACS PLUS: ACS/OBSS, ICM, UMAC, 80 MHz Scan, Dynamic Channel Selection-Interference Mitigation (DCS-IM) for 802.11n and 802.11a, and enhanced channel switch and channel scan API.

<a href="#">Chapter 15, Spectral Scan and Analysis</a>	Explains the essential components of the spectral scan and analysis feature, including an overview of system functionality, configuration parameters, reporting formats, host driver software architecture, host-firmware interaction (for 11ac chipsets), and supporting applications and tools.
<a href="#">Chapter 16, Band Steering Methods</a>	Discusses the band steering approaches, such as Transmit Beamforming (TxBF), Single AP Band Steering, AP steering of legacy clients, band steering in QWRAP mode, Multi-AP Coordinated Steering and Adaptive Path Selection, and SSID Steering.
<a href="#">Chapter 17, Repeater Access Point Functions</a>	Describes the functions of an access point (AP) that works as a range extender (RE) or a repeater, such as Dual-Band Dual-Concurrent (DBDC) repeater, extender APs, Qualcomm Wireless Repeater Access Point (QWRAP), Power Amplifier PRe-Distortion (PAPRD), Wireless Distribution System (WDS), Proxy STA, and Wi-Fi Protected Setup (WPS) for range extenders and repeaters.
<a href="#">Chapter 18, Voice and Multimedia Features</a>	Describes the capabilities supported for voice and multimedia traffic, such as Wi-Fi Multimedia (WMM) Flow Control and Buffer Management, video over wireless, voice enterprise, Smart Antenna, and Unified Smart Antenna API.
<a href="#">Chapter 19, Wi-Fi SON Features</a>	Explains the Wi-Fi Self Organizing Network (SON) capabilities, such as Range Extender Placement and Auto-configuration (repacd), Daisy Chain and Best Uplink Selection, Multi-SSID and Traffic Separation, and the Powerline Communications (PLC) integration with Wi-Fi SON.
<a href="#">Chapter 20, Coexistence of WLAN with NoDS mesh, BT, ZigBee, and CSRmesh</a>	Discusses the NoDistribution System (NoDS) frame support for mesh networks. Also, this chapter describes the coexistence of WLAN with Bluetooth (BT), ZigBee protocols, and CSRmesh devices.
<a href="#">Chapter 21, WiGig (802.11ad Standard) Features</a>	Describes the IEEE 802.11ad specification-compatible functionalities, which enable operations on 60 GHz frequency bands.
<a href="#">Chapter 22, Logging and Statistics for Debugging</a>	Describes the utilities that can be used for diagnostic and troubleshooting purposes, such as packet logs, commands to display AP statistics, and WLAN event reports.

**NOTE** All 160/80+80 MHz related information is applicable for the QCA9994 and QCA9888 platforms only.

## Terms and Acronyms

The following table lists the terms and acronyms that are used in this document:

Acronym	Definition
ACL	Access Control List
ACS	Auto Channel Selection
AP	Access Point
ASF	Atheros Service Framework
DBDC	Dual Band Dual Concurrent
DFS	Dynamic Frequency Selection
EAP	Extensible Authentication Protocol
FTM	Fine Timing Measurement
FTMRR	Fine Timing Measurement Range Request
HAL	Hardware Abstraction Layer
HIF	Host Interface
HT	High Throughout
HTC	Host Target Communications
LAN	Local Area Network
LCI	Location Civic Information
LCR	Location Civic Report
LMAC	Lower MAC
LOWI	Location Wi-Fi Interface
MAC	Medium Access Control
P2P	Peer to Peer
QDF	Qualcomm Driver Framework
RIFS	Reduced Inter Frame Spacing
STBC	Space-Time Block Codes
TxBF	Transmit Beamforming
UAPSD	Unscheduled Automatic Power Save Delivery
UMAC	Upper MAC
WDS	Wireless Distribution System
WLAN	Wireless LAN
WMI	Wireless Message Interface
WMM	Wi-Fi Multimedia
WMM-AC	WMM admission control
WPA	Wi-Fi Protected Access

## Additional Resources

Qualcomm Technologies Reference Design hardware, software, and documentation contain proprietary information of Qualcomm Technologies, Inc., and are provided under a license agreement containing restrictions on use and disclosure, and are also protected by copyright law. Reverse engineering of this hardware, software, or documentation is prohibited.

This guide assumes that the reader has studied and is familiar with the following:

- IEEE 802.11 standards
- C Programming language

The following resources should be referenced regarding topics that are not addressed in this document:

- Software release notes
- Product-specific Setup Guides (describes the software image installation process)
- Driver Build Guides (include platform-specific and porting information)

See [Revision history](#) for an indication of content that has been added or changed in this revision of the document.

QUALCOMM  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

# 1 WLAN AP Driver Architecture

---

The major components of the Qualcomm Technologies AP software are Wireless LAN, Ethernet, Router Stack, and Hybrid Network. This document focuses on the Wireless LAN component.

## 1.1 Wireless LAN

The Wireless LAN (WLAN) component is one of the major components of the AP system software. Typically AP platforms would have one WLAN circuit for a single-band product; Dual Band Dual Concurrent (DBDC) AP platforms would have two WLAN circuits. These WLAN devices could be configured with legacy 802.11 a/b/g standards, or as single/two/three streams 802.11n devices. WLAN software is responsible for controlling all these WLAN devices to provide WLAN services for the AP platform.

WLAN software is classified into the following two major subcomponents:

- WLAN driver
- WLAN application

The WLAN driver is the core of the WLAN software. It implements various 802.11 standards and provides WLAN services for the AP. The WLAN application includes various tools for configuring and debugging the WLAN driver. Also included are hostapd, supplicant for 802.1X/WPA/WPA2/EAP authenticators.

A significant amount of CPU power is required due to the complexity of WLAN software. However, some platforms have a limited amount of host CPU power. It is therefore sometimes necessary to offload the WLAN software to the target CPU for these platforms. The target CPU is a microprocessor that is integrated into SoC WLAN chipsets. Based on the AP hardware platform and host CPU offload requirements, the WLAN software could be integrated into AP software using three different models:

- Direct Attach Model
- Full Offload Model
- Partial Offload Model

**NOTE** For some of the Qualcomm Technologies products, the command line configuration utility has moved to UCI. Therefore, for all the configuration (cfg) commands in this document, refer to the corresponding UCI commands except the ones that are marked as ‘internal use/debugging use only’, which continue to support cfg.

### 1.1.1 Direct Attach Architecture

In this model, the entire WLAN software runs on the host and interfaces with the WLAN hardware through the PCIe or AXI bus (see [Figure 1-1](#)).

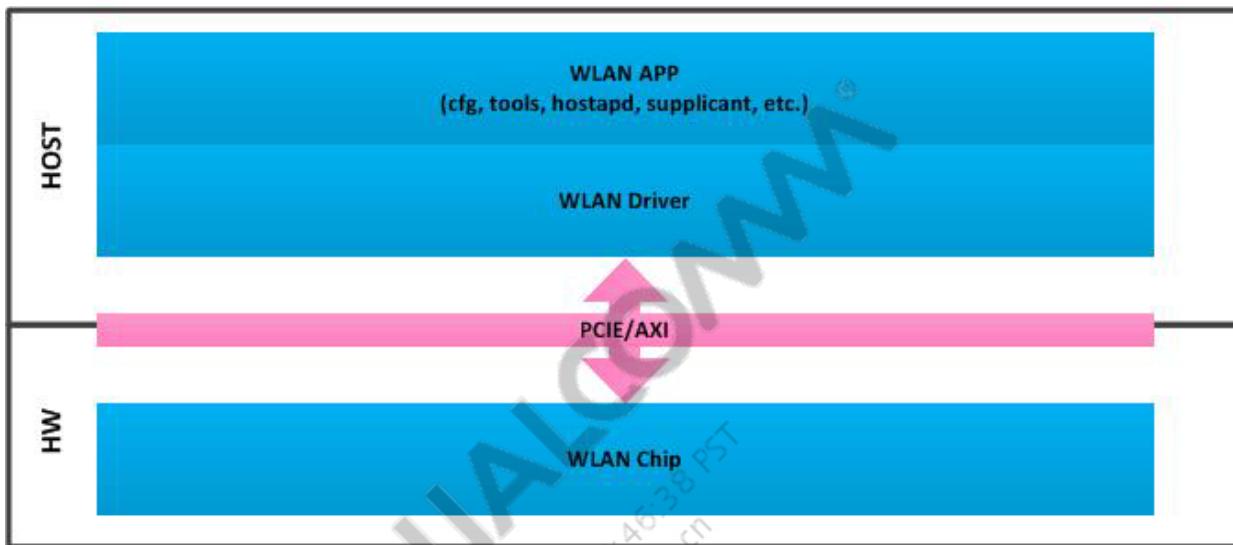


Figure 1-1 Direct Attach Architecture

### 1.1.2 Full Offload Architecture

In this model, some of the WLAN driver components and the hostapd/supplicant from the WLAN application component run on the host and the remaining application subcomponents of the WLAN application run on the target. A thin target interface layer is added on the host and a thin host interface layer is added on the target for the host-target communications. The hardware interface between the host and target may be a USB, MII, PCIe or AXI interface.

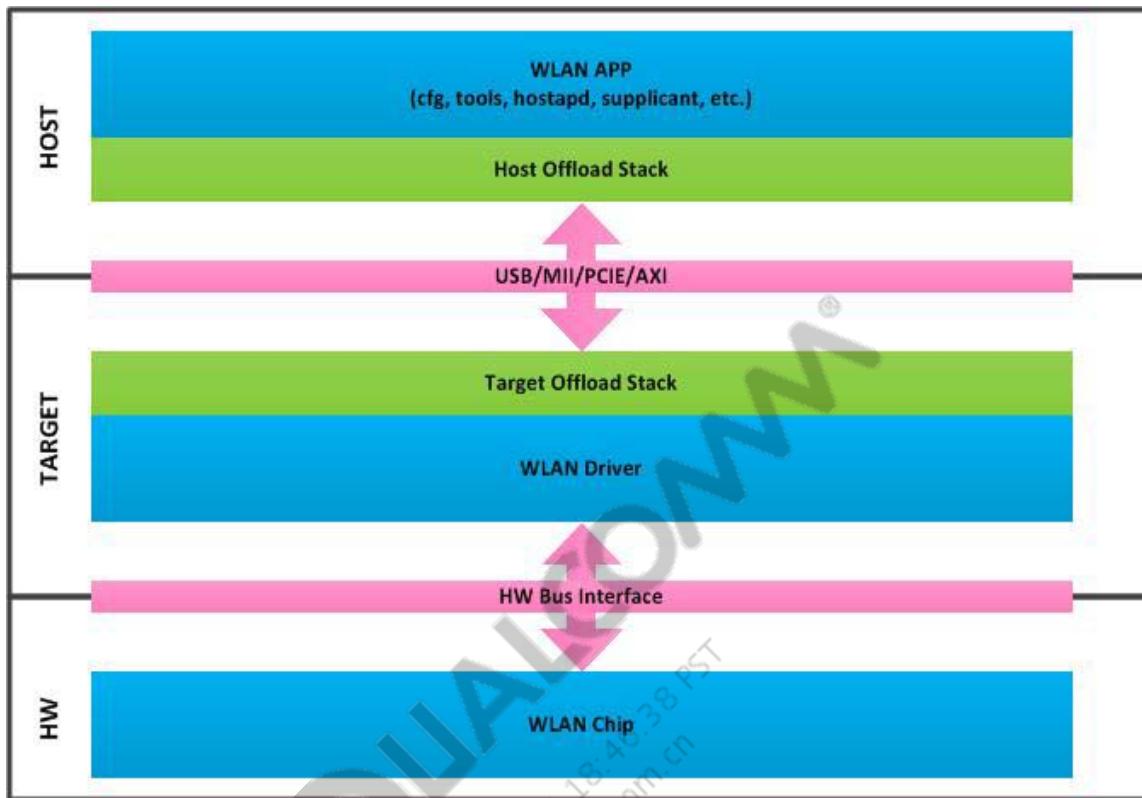


Figure 1-2 Full Offload Architecture

## 1.2 WLAN Software Architecture

The WLAN driver is implemented as well-defined layers, and each layer has well defined APIs. This enables the customer to customize the AP software and to replace some of these layers with their own implementations.

Figure 1-3 illustrates the overall architecture of the WLAN software

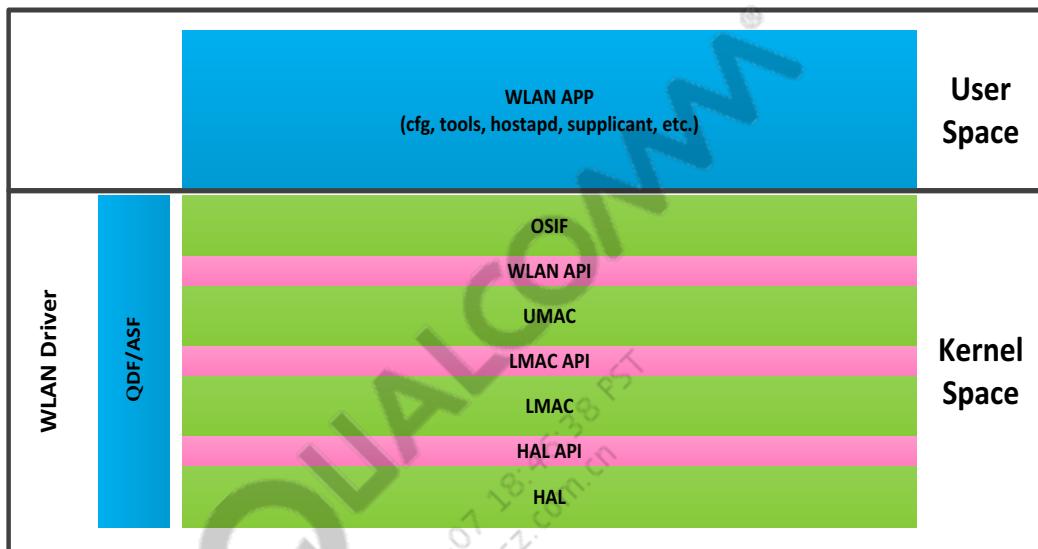


Figure 1-3 WLAN Software Layers

### 1.2.1 Hardware Abstraction Layer (HAL)

The Hardware Abstraction Layer (HAL) is the section of code that interfaces the driver to the specific chipset being used. If multiple chips are being used, such as in a dual concurrent AP configuration, then an instance of HAL is created for each chip. Thus, it is quite easy to mix and match various Qualcomm Technologies chipsets in any particular design.

The HAL code is split into two major areas; common HAL interface code that is used by the upper layers to communicate with the HAL, and chip-specific HAL modules that support the particular hardware implementation. There are currently three main HAL chip specific branches, called AR5212, AR5416, and AR9300. The modules are described as follows:

AR5212	This module supports the legacy devices that cover the 802.11b/g/a mode operations. These devices often had separate PHY chipsets that could be matched with the MAC chip, so the support for various PHY chipsets are also included in this module.
AR5416	This module supports the 1st, 2nd, and 3rd generation of IEEE 802.11n-capable chipsets. These include the 1st generation AR5416 chipset that is used with an 11n capable PHY chip, 2nd generation AR91xx, and the 3rd generation integrated 92xx series chipsets that combine the PHY and MAC into a single chip. Also, the 1x1 AR958x chipsets are supported by this module
AR9300	The AR93xx/AR958x series chipsets are supported by this module. These chipsets use a new advanced radio design, a new MAC queuing interface, and support 3-stream MIMO operations.

The HAL jump interface is an abstraction layer that tends to be as generic as possible to be reused on multiple OSs and for multiple chip families. Therefore, it is important to only call functions that are within the HAL module from within the HAL; HAL should not have any linkage to external routines. Hence, the `ah_hal_printf` function is used instead of the general `printf` functions, and the `HDPRINTF` macro is used instead of the more common `DPRINTF` macro used in the rest of the driver.

## 1.2.2 HAL API

The HAL exports a set of unified APIs for the LMAC to access HAL. These APIs are common for different chip-specific HALs.

## 1.2.3 Lower MAC (LMAC)

The Lower MAC portion consists of the Atheros Device Object (ATH). The ATH layer is responsible for managing the data flow into the input queues of the hardware, as well as managing lower layer protocols, such as Block ACK processing. The design of the LMAC is driven by Qualcomm component hardware architecture vice the UMAC, which is more generally driven by the IEEE 802.11 protocol specifications.

The LMAC provides the following major features, as controlled by the UMAC layer and the OSIF requirements:

- Unified transmit and receive path for both legacy and 11n chipsets including Buffer and Queue management
- Rate control, DFS, spectral scanning
- Advanced 11n MAC features such as aggregation, RIFS, MIMO power save
- 802.11 network power save and device power state (D0-D3) management
- Beacon generation and TSF management
- Wake-On-Wireless support
- RfKill, Customized LED and GPIO algorithms
- Provides support for value added features such as IQUE, WoW, Smart Antenna, TxBF

### 1.2.4 LMAC API

The LMAC exports standardized APIs for the UMAC to access the LMAC. This enables customers to replace our UMAC with their own UMAC layer. It is therefore important for the UMAC to access the LMAC through these APIs, and any direct access to LMAC data structures from the UMAC should be avoided.

### 1.2.5 Upper MAC (UMAC)

The Upper MAC is the portion of the MAC that performs most of the 802.11 protocol processing, and provides the interface to the OS interface layer. The major functionalities of the module include the following:

- AP state machine
- Connection state machine and node management
- MLME services
- Basic features such as scanning, encryption, power management, regulatory domain, resource management
- Various protocol specific features such as P2P, WMM-AC, RRM
- Various infrastructure supports such as WDS and EXTAP
- Provides support for value added features such as ACL, ACS, IQUE, VOW, smart antenna, TxBF

### 1.2.6 WLAN API

UMAC exports standardized APIs for the OSIF layer to access the UMAC. It is important for OSIF to access UMAC through these APIs and any direct access to UMAC data structures or UMAC function calls from UMAC should be avoided.

### 1.2.7 OS Interface Layer (OSIF)

The OS interface layer is an OS-specific module that implements a network stack interface and application-specific interfaces for the WLAN driver. This module uses WLAN APIs provided by the UMAC to implement these interface functions. Each OS should have its own implementation of this OS interface layer.

### 1.2.8 Qualcomm Driver Framework (QDF)

This section of the code provides a unified interface for all driver-related functions to make porting to other operating systems easier. This is an abstraction layer that is called from the driver for functions like creating timers and tasklets. This abstraction layer can be modified to support whichever operating system is chosen to support. Implemented support is provided for Linux 2.6 OS, which is the subject of this guide.

Any hardware specific or OS-specific items that do not belong in the WLAN OS interface layer should be placed in the QDF. When exporting a new interface, be sure to include it in the QDF files.

## 1.2.9 Atheros Service Framework (ASF)

This section of the code provides a unified interface for all basic service related functions to make porting to other operating systems easier. This layer provides abstraction for basic services like memory interface functions, debug related functions, and so on. This abstraction layer can be modified to support whichever operating system you choose to support. Implemented support is provided for Linux 2.6 OS, which is the subject of this guide.

## 1.2.10 WLAN Application

Wireless LAN applications run in the user context and includes

- **Wireless tools:** for all WLAN driver configurations
- **apcfg:** configuration of AP platform.
- **hostapd:** 802.1X/WPA/WPA2/EAP Authenticator for AP mode of operations including WPS support.
- **wpa\_supplicant:** 802.1X/WPA/WPA2/EAP Authenticator for STA mode of operation including WPS support.
- **radartool:** tool to test/debug radar detection.
- **Spectral daemon**
- **spectraltool:** tool to configure/debug spectral scan.
- **athssd:** demo tool to carry out Spectral Scan and Analysis, and output interference information
- **pktlog**

## 1.2.11 Unified Software WLAN Architecture

This section provides an overview of the unified software stack with its layers and components, followed by a description of the enhancements to be applied to the stack to be VHT compliant for both the AP and STA modes of operation. The unified stack supports all QCA solutions – both direct attach and offload solutions.

### 1.2.11.1 Offload stack

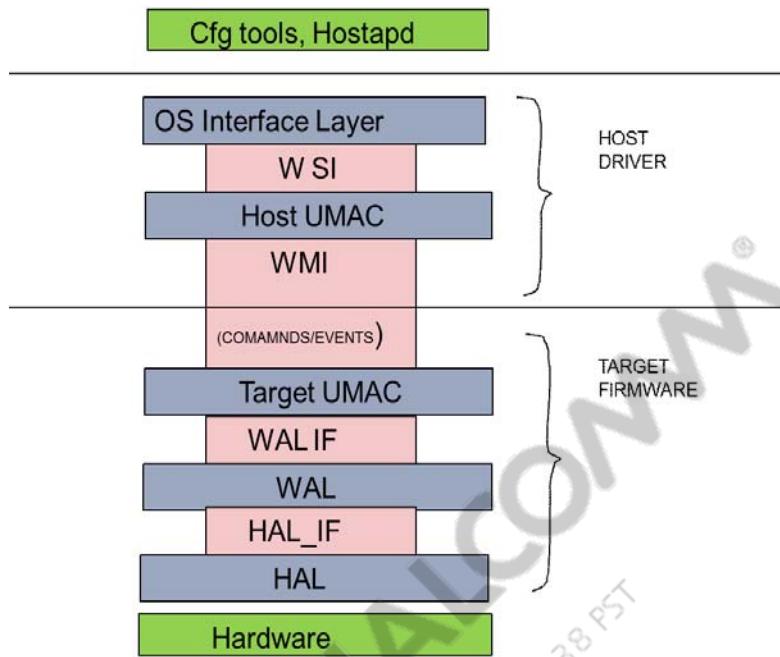


Figure 1-4 Offload stack diagram

- **OS Interface Layer:** Implements the network stack and application specific interfaces for the WLAN driver.
- **WSI:** WLAN Service API consists of functions exported by the UMAC for OSIF access (avoids direct access to UMAC data structures)
- **UMAC:** Most of the 802.11 protocol specifics are implemented at the Upper MAC layer (AP state machine, connection state machine, MLME services)
- **Target UMAC:** Handles lower level MAC functions. Manages data flows (TX and RX) to the input queues of the hardware, buffer management, rate control, Aggregation, MIMO power save.
- **WAL:** Abstracts higher level features of different MACs.
- **HAL:** Hardware Abstraction Layer for low level hardware functions such as descriptor setup, encryption key setups, channel setup.

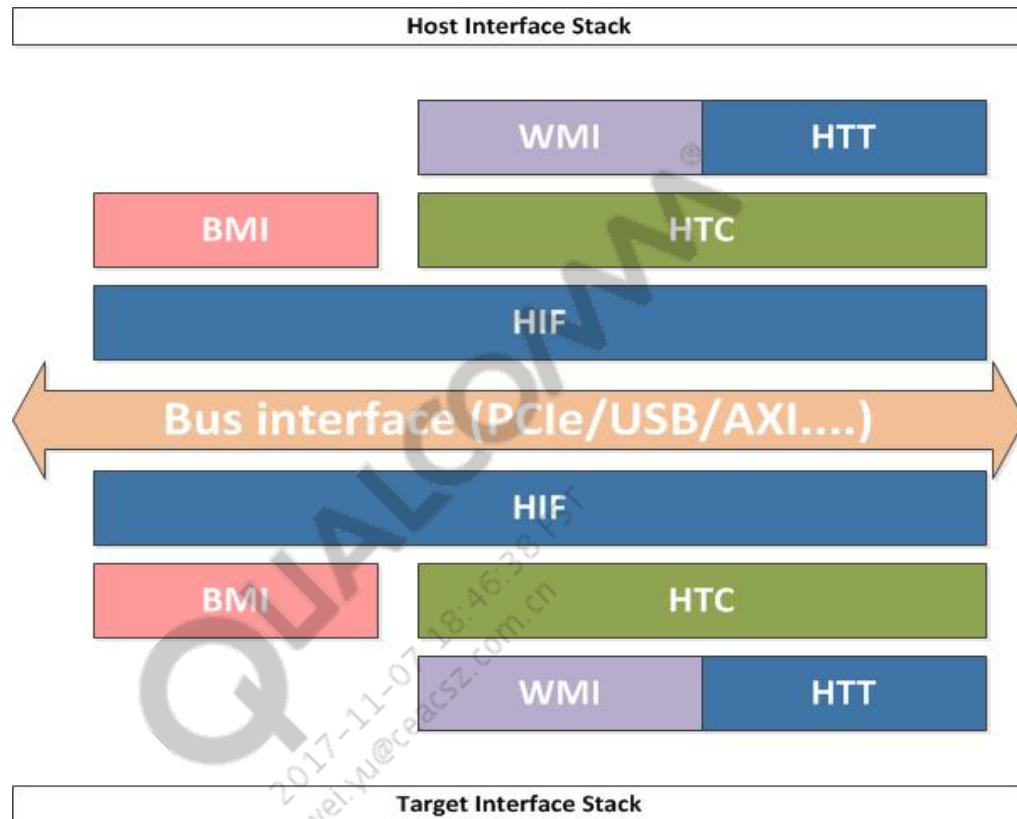
#### Unified UMAC support

- The new stack supports both direct attach and offload solutions by using indirect function pointer-based dynamic linking.
- For direct attach solutions, the old IF\_LMAC function pointers will be used while for the offload stack the OL\_WMI function pointers will be used.
- Set of UMAC functions are optionally enabled/disabled to run on the host or target.

Given that the existing architecture is modular, the new offload WMI layer can be easily plugged in.

### 1.2.11.2 11AC offload host-target interface

Host-to-target communication happens through a well-defined software interface layers. This section describes these software interface layers.



**Figure 1-5 Host-to-Target interface diagram**

#### Host interface layer (HIF)

This layer abstracts the bus interface between the host and target, and provides a communication mechanism between the host and target.

#### Bootloader Message Interface (BMI)

During initialization, the host has an opportunity to use the bootloader message interface (BMI) services that allow the host to get target revision information, download (write) code and data into specific locations in target memory, upload (read) code and data from target memory, read/write target registers, and execute at a specific target address. The BMI downloads special applications, tests, and patches to code and to data, and sometimes sets global variables in the target application.

#### Host Target communication Layer (HTC):

HTC is the message transport used by WMI and HTT to send and receive control and data messages across the interconnect that connect the 11ac device target and the host system.

### Wireless Message Interface (WMI)

Host-to-target control path communications are done through the WMI interface. A set of well-defined WMI messages are used by the host WLAN stack to communicate with the target WLAN stack and vice versa.

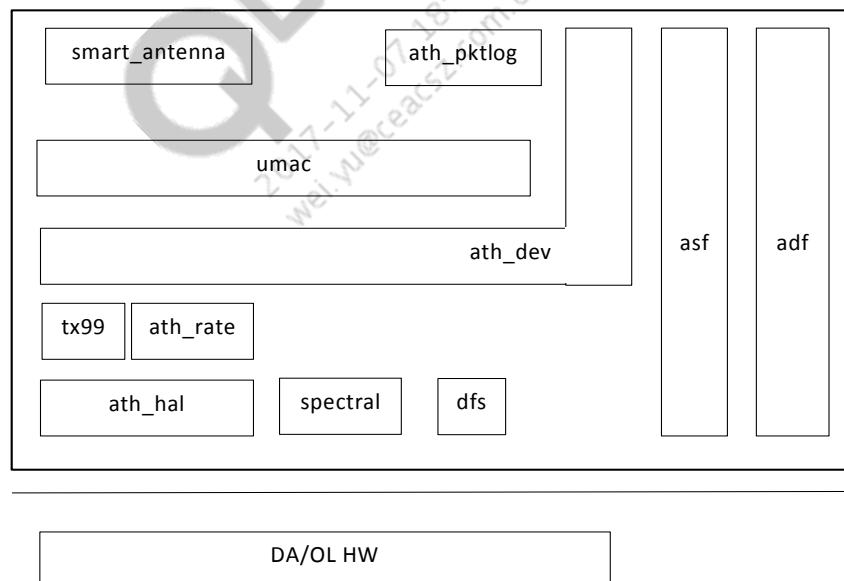
### Host Target Transport Layer (HTT)

Host-to-target data path communications are done through the HTT interface. A set of well-defined HTT messages are used by the host WLAN stack to communicate with target WLAN stack and vice versa.

## 1.3 WLAN driver modules architecture until QCA\_Networking\_2016.SPF.2.0 release

This section describes the WLAN Driver kernel modules of both Direct-Attach and Off-load chipsets.

The following is the WLAN driver kernel modules block diagram.



**Figure 1-6 Driver - Block Diagram until QCA\_Networking\_2016.SPF.2.0 release**

The WLAN Driver modules consists of the following Kernel Objects.

1. asf.ko – Basic Framework module
2. qdf.ko – Basic Framework module
3. ath\_spectral.ko – Spectral Support
4. ath\_dfs.ko – DFS support

5. umac.ko – Common 802.11 protocol Management
6. ath\_hal.ko – Direct-Attach HW abstraction Layer
7. ath\_rate\_atheros.ko – Direct-Attach Rate Control Support
8. hst\_tx99.ko – Direct-Attach Tx99 Support
9. ath\_dev.ko – Direct-Attach LMAC Layer
10. qca\_da.ko – Direct-Attach Driver Support
11. qca.ol.ko – Offload Driver Support
12. smart\_antenna.ko – Smart Antenna Support
13. ath\_pktlog.ko – Direct-Attach Packet logging Support

### 1.3.1 Common WLAN Driver Modules

The Common WLAN Driver Modules are used for both Direct-Attach and Off-load Chipsets.

The asf.ko, qdf.ko, ath\_dfs.ko, ath\_spectral.ko and umac.ko are needed for both Direct-Attach and Off-load Chipsets.

### 1.3.2 Direct-Attach specific WLAN Driver Modules

These modules are needed only on platforms with Direct-Attach chipsets

ath\_hal.ko, ath\_rate\_atheros.ko, hst\_tx99.ko, ath\_dev.ko, qca\_da.ko

They can be used after umac.ko is inserted. qca\_da.ko has the kernel PCI subsystem interface. The Direct-Attach chipsets are detected only after loading qca\_da.ko.

### 1.3.3 Offload specific WLAN Driver Modules

qca.ol.ko has support for Offload chipsets. This module can be used after umac.ko is inserted to detect Offload radios. qca.ol.ko also has the support for NSS Wifi-Offload feature.

## 1.4 Offload and direct-attach driver modularization

This section describes the design and implementation of the modularization of the existing WLAN driver.

As the WLAN driver evolved, the modularity of the driver faded. Particularly, when the Offload (OL) support was added, all the OL related code is part of the UMAC module. As long as both radios (OL and DA) are used at all times, there is no specific issue with this, but when either OL or DA (Direct-Attach) radio is used, then it would benefit if only the particular driver (either OL or DA) is used, instead of having all of the WLAN Driver.

With the new QCA chipsets not being based on DA model, it is reasonable to have common kernel modules and separate DA and OL modules. This way, the common modules will be used for all QCA Radios and depending on which Radio the platform uses, the corresponding DA or OL modules will be used.

This architecture also applies for the future QCA chipsets, which may not follow either DA or OL model. The following are the design issues with the WLAN driver until QCA\_Networking\_2016.SPF.2.0 release:

- OL code is part of UMAC and dependent on DA modules (LMAC, HAL).
- UMAC modules contains code that should have been modularized into OL and DA specific modules.

### Driver - Block Diagrams

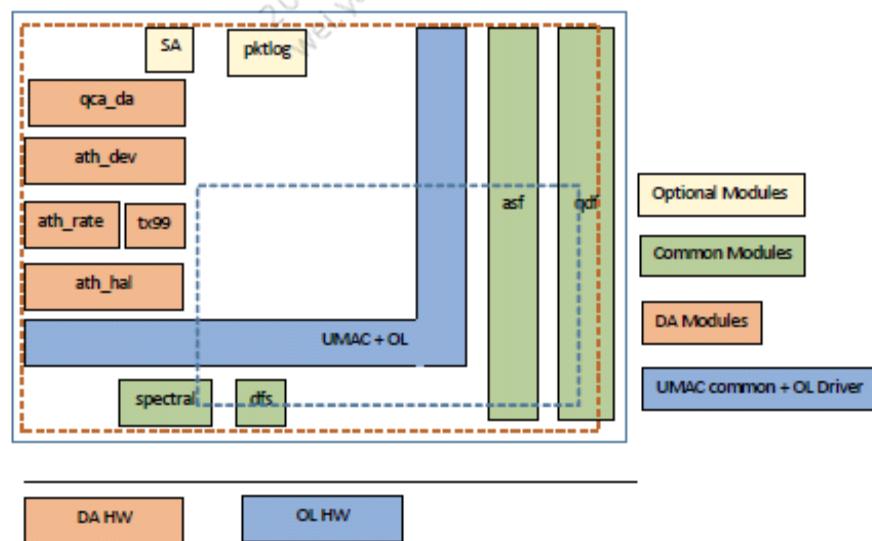
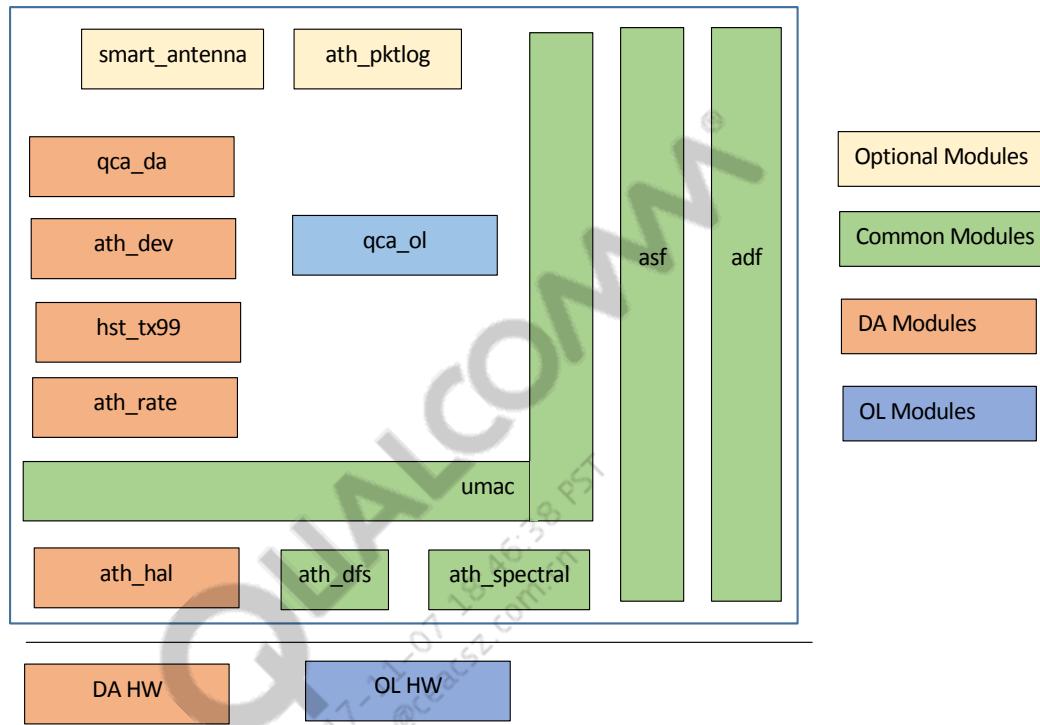


Figure 1-7 Driver - Block Diagram until QCA\_Networking\_2016.SPF.3.0 release

As shown in the above block diagram, all modules are used for DA or OL HW. This approach is good, if all platforms have both DA and OL HW. But if platforms have either of these, then the size of the driver is not optimized for the specific radio. A customer using either DA or OL radio is forced to use the complete driver, which is currently about 4 MB in size.



**Figure 1-8 Block Diagram in QCA\_Networking\_2016.SPF.4.0 release and later**

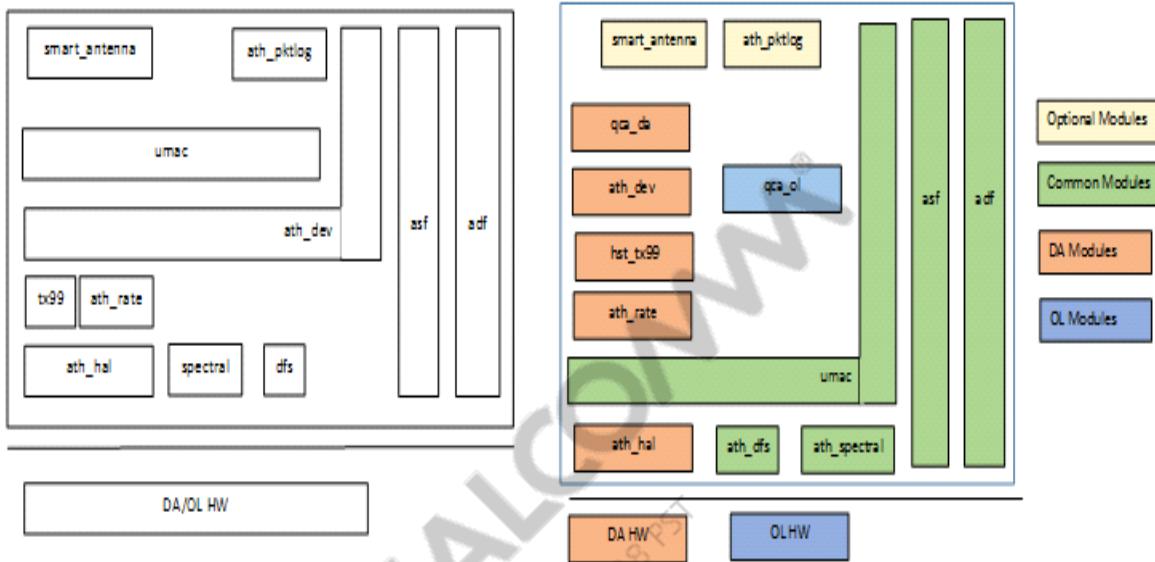
The preceding figure shows the proposed dependencies of different driver modules after the Modularization is implemented.

The following points must be considered:

- The asf and adf modules are part of core framework and they are always needed.
- ath\_dfs and ath\_spectral modules are common to both OL and DA chipsets, if support is not needed, they can be disabled at compile time.
- The umac module is also common for both OL and DA, which implements part of the protocol and Data-path at the Host layer.
- The optional modules are used depending on the needed support.
- The other modules are used depending on the type of HW.
  - The DA HW will use HAL, rate, LMAC(ath\_dev) and the qca\_da module.
  - The OL HW will use qca\_ol module.
- The qca\_da and qca\_ol modules implement the HW specific functions and are independent of each other.

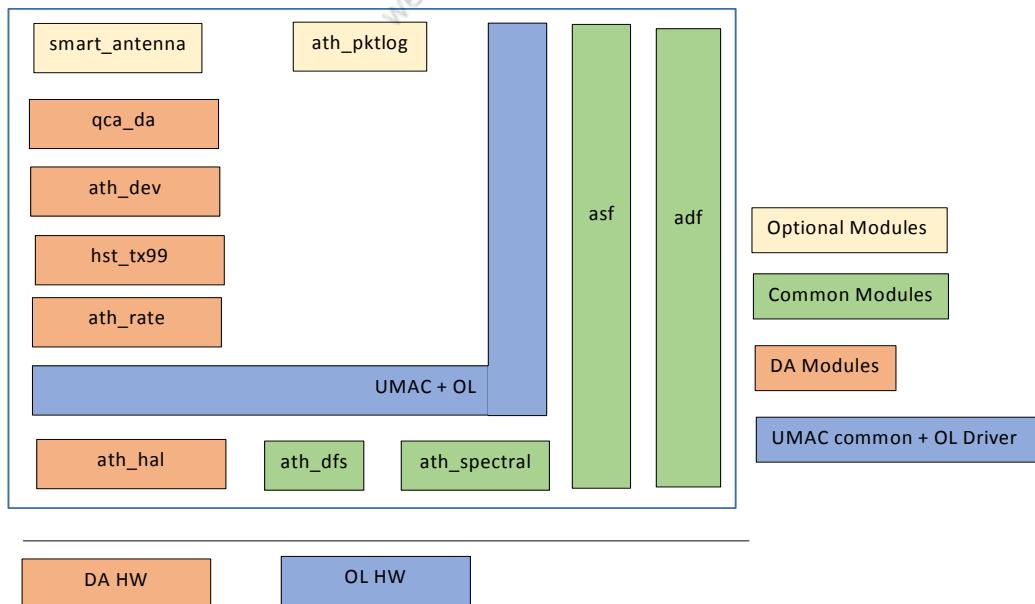
## Block Diagram Comparison

Old Block Diagram    Vs    New Block Diagram



**Figure 1-9 Block Diagram Comparison**

## WLAN driver block diagram



**Figure 1-10 Driver - Block Diagram in QCA\_Networking\_2016.SPF.3.0**

The preceding diagram shows the dependency of the WLAN driver modules for QCA\_Networking\_2016.SPF.3.0

The following points need to be noted:

- The asf and adf modules are part of core framework and they are always needed.
- The spectral and dfs modules are used as it is, i.e they are required for both OL and DA.
- The OL module will be part of UMAC, as it is currently.
- All DA module dependency of the UMAC will be removed from the UMAC.
- All DA specific functions within the UMAC will be moved to qca\_da module.
- DA driver will have dependency on the OL driver.

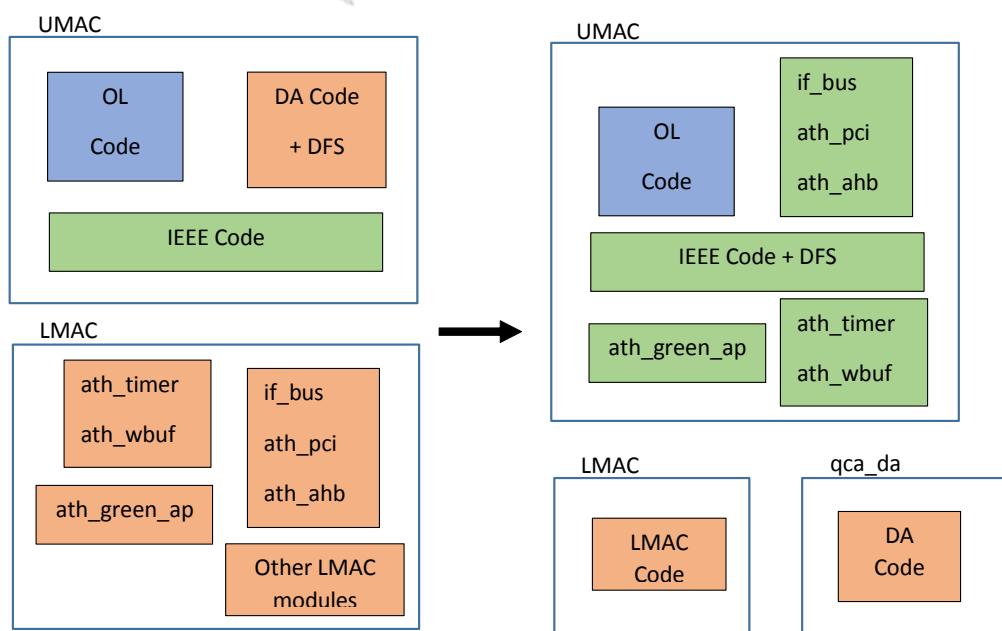
### WLAN driver modularization

In Phase 1 (QCA\_Networking\_2016.SPF.3.0 release), the OL module is made independent of the DA module. The following are the major changes implemented:

- Common LMAC (ath\_dev) functions (non-DA specific) to move to UMAC.
- DA specific functions in UMAC to move to qca\_da module.
- DA functions in UMAC to be made common and moved to ieee80211 layer

Following in the diagram depicting the above changes and later described below.

**NOTE** The diagram doesn't show all the blocks of the modules. Only the blocks that involve changes are shown.



**Figure 1-11 Phase 1 Changes**

## L\_MAC (ath\_dev) functions to move to UMAC

Since the UMAC should be made independent of the L\_MAC, some functions need to be moved to UMAC. These functions are not DA specific and can be safely moved to UMAC. They also need to be moved to satisfy OL driver dependency.

Following are the files to move to UMAC module.

- ath\_timer
- ath\_green\_ap
- ath\_wbuf
- if\_bus
- ath\_pci
- ath\_ahb

Following functions will be made common and moved to ieee80211 layer, so that they can be directly called from OL Layer.

- ath\_new\_opmode -> ieee80211\_new\_opmode
- ath\_vap\_iter\_new\_opmode -> ieee80211\_vap\_iter\_new\_opmode

## DA functions in UMAC to move to qca\_da

All DA specific functions in the UMAC will be moved to a new module qca\_da. This module, along with ath\_hal, ath\_rate, ath\_dev, tx99 will be used only if DA HW is present.

Following are the DA files to move to qca\_da module.

- ath\_linux
- ieee80211\_aponly
- if\_ath
- if\_ath\_amsdu
- ath\_cwm
- ath\_cwm\_project
- if\_ath\_uapsd
- if\_ath\_dfs
- if\_ath\_aow
- if\_ath\_quiet
- if\_ath\_mat

## DA functions in UMAC to be made common and moved to ieee80211 layer

These functions are currently present in DA specific layer of the UMAC (if\_lmac layer), but these functions are used by OL layer too. Because if\_lmac layer will move to qca\_da module, these functions will be moved to the ieee80211 layer.

- Following are the functions that will be moved to ieee80211 layer.
- find\_alternate\_mode\_channel
- ieee80211\_random\_channel
- ieee80211\_get\_test\_mute\_chan
- ieee80211\_vap\_iter\_update\_bss\_chan
- ieee80211\_dfs\_action
- ieee80211\_mark\_dfs
- ath\_net80211\_bufffull\_handler renamed to ieee80211\_bufffull\_handler.  
ic\_notify\_bufffull will be removed.
- ath\_net80211\_get\_ald\_statistics renamed to ieee80211\_get\_ald\_statistics.  
ic\_get\_ald\_statistics will be removed.
- ath\_net80211\_add\_hmmc renamed to ieee80211\_add\_hmmc .  
ic\_add\_hmmc will be removed
- ath\_net80211\_del\_hmmc renamed to ieee80211\_del\_hmm  
ic\_del\_hmmc will be removed

### **Kernel PCI/AHB Interface**

The PCI device registration will be done independently for OL and DA HW. The UMAC module will register only OL devices. The qca\_da module will register only DA devices. Each will have its own PCI probe functions and will initialize devices independently.

This is also true for AHB devices, that are OL and DA based. For example, QCA9558 is DA based and IPQ4018 is OL based.

## **1.5 Additional changes for WLAN driver modularization**

Starting with the QCA\_Networking\_2016.SPF.4.0 release, both OL and DA drivers are made independent and a UMAC module is created, which is common and independent of both OL and DA modules.

Because the DA driver is already independent of ‘UMAC+OL’ driver, splitting the UMAC and the OL modules into 2 separate modules is implemented.

The following high-level design enhancements are made:

- Put together all the OL files into a new module (qca.ol). This module will be dependent on UMAC module.
- Create an independent UMAC modules, which is independent of OL and DA modules.

### **Identifying Offload specific files**

The current UMAC module has both common and Offload functions and files.

## Current Offload files

As all Offload files are already part of a separate “offload” directory in the WLAN driver source code, these files are put together to form the new “qca\_ol” module.

## Kernel PCI/AHB Interface files

With separated the DA device registration process, the current PCI and AHB initialization files that are part of the UMAC module are moved to the new qca\_ol module.

## Independent UMAC module

The goal of an Independent UMAC module, is to have all common code between Offload and Direct-Attach under one module, such that Offload and Direct-Attach modules can make use of this common code independently. Having common code in one module will also reduce the size of the Driver.

As all offload files are moved to the qca\_ol module, any UMAC files that directly calls offload functions will be categorized into one of the following options and resolved accordingly.

- OL parameter function calls from UMAC
- OL functions bypassing “ic” pointers to enter OL layer
- Functions in UMAC, but are OL specific
- Functions independent of OL, but present in OL layer
- UMAC functions used by OL and DA modules
- NSS (WiFi OL) modularization

## OL parameter function calls from UMAC

There are quite a few OL functions which are meant to configure OL driver parameters, that are directly called from the UMAC.

As an independent UMAC, cannot make a direct function call to OL layer, these functions should use callback functions to enter OL or DA layer.

As “ic\_vap\_get\_param” is not available, this function will be added.

For “set” functions, “ic\_vap\_set\_param” will be used.

For example:

```
ol_txrx_clear_rawmode_pkt_sim_stats
ol_txrx_host_stats_get
ol_tx_rst_tso_stats
ol_txrx_print_rawmode_pkt_sim_stats
ol_ath_set_vap_cts2self_prot_dtim_bcn
ol_tx_rst_sg_stats
ol_tx_print_sg_stats
ol_rst_rx_cksum_stats
ol_tx_print_tso_stats
ol_txrx_host_msdu_ttl_stats
ol_txrx_debug
```

```

ol_txrx_fw_stats_get
ol_ath_ucfg_reset_peer_mumimo_tx_count
ol_txrx_aggr_cfg
ol_rate_is_valid_basic
ol_ath_net80211_get_vap_stats
ol_txrx_host_me_stats
ol_txrx_fw_stats_cfg
ol_txrx_host_stats_clr
ol_print_rx_cksum_stats

```

### **OL functions bypassing “ic” pointers to enter OL layer**

All functions to the OL or DA layer, from the UMAC should be done only as a callback, i.e using the “ic” function pointers.

So any function that is found to have bypassed this protocol, will be assessed and will be either called through “ic” pointer, or the whole block will be moved to OL layer.

The changes to call Via the “ic” pointers will be broadly the following.

1. Add a function pointer to “struct ieee80211com”
2. Register (Initialize the function pointer) the OL function, during OL device attach.
3. Call the function pointer when needed to enter OL layer.

For example:

```

ol_ll_pdev_tx_lock
ol_ll_pdev_tx_unlock
ol_txrx_osif_vdev_register
ol_tx_tso_sg_process_skb
ol_ath_ucfg_get_peer_mumimo_tx_count
ol_net80211_set_mu_whtlist

```

### **Functions in UMAC, but are OL specific**

A lot of Data path functions specific to OL layer are part of the UMAC module in the osif\_umac file. Since these files are only for the OL devices, these will be moved to the offload module itself.

For Example:

```

osif.ol.ll.vap.hardstart
osif.ol.hardstart.vap.vow.debug
osif.deliver_data.ol

```

### **Functions independent of OL, but present in OL layer**

There are also functions within the OL layer, that are independent of OL layer, which can be made common and part of the UMAC module. These functions will be moved to the UMAC module.

For Example:

```

transcap_nwifi_to_8023
dscp_tid_map

```

## UMAC functions used by OL and DA modules

As UMAC is inserted before the OL or DA module, the functions that are needed by OL or DA modules, just need to be exported.

A new file umac\_exports.c was added during Phase 1 development, this file will be used to export the necessary functions.

## NSS (WiFi OL) modularization

The osif\_nss files will be part of the ‘umac’ module, but osif\_nss\_wifiol related files will be moved to the ‘qca\_ol’ module, as these are specific to the OL chipsets.

This puts a limitation that, ‘umac’ calls to osif\_nss\_wifiol functions cannot be made directly, as the ‘umac’ module will be inserted before the ‘qca\_ol’ module.

To overcome this limitation, the osif\_nss.ol\_pdev\_attach function, will be overloaded with a pointer to an array of function pointers, which will be initialized to the respective osif\_nss\_wifiol functions.

Following is the structure nss\_wifi\_offload\_funcs which has the array of pointers initialized to the respective functions. This structure will be passed to ic->nss\_funcs in osif\_nss.ol\_pdev\_attach function.

```
struct nss_wifi_offload_funcs nss_wifi_funcs = {
    osif_nss.ol_store_other_pdev_stavap,
    osif_nss.vdev_me_reset_snooplist,
    osif_nss.vdev_me_update_member_list,
    osif_nss.ol_vap_xmit,
    osif_nss.vdev_me_update_hifitlb,
    osif_nss.vdev_me_dump_denylist,
    osif_nss.vdev_me_add_deny_member,
    osif_nss.ol_vdev_set_cfg,
    osif_nss.vdev_process_mpsta_tx,
    osif_nss.ol_wifi_monitor_set_filter,
    osif_nss.vdev_get_nss_id,
    osif_nss.vdev_process_extap_tx,
    osif_nss.vdev_me_dump_snooplist,
    osif_nss.ol_vap_delete,
    osif_nss.vdev_me_add_member_list,
    osif_nss.vdev_vow_dbg_cfg,
    osif_nss.ol_enable_dbdc_process,
    osif_nss.vdev_get_nss_wifiol_ctx,
    osif_nss.vdev_me_delete_grp_list,
    osif_nss.vdev_me_create_grp_list,
    osif_nss.vdev_me_delete_deny_list,
    osif_nss.vdev_me_remove_member_list
};
```

## “qcawifi.sh” script changes

The qcawifi.sh script, which inserts the WLAN driver modules and passes the module parameters also will need changes due to this modularization.

## Module Parameter split between UMAC and OL modules

All module parameters are currently passed to the UMAC module. Due to the split between qca\_ol and qca\_da modules, the qcawifi.sh script should be changed to pass the corresponding module parameter to the correct module.

Following are the module params passed to the corresponding modules.

module_param	module_name
enableuartprint	qca_ol.ko
enable_tx_tcp_cksum	qca_ol.ko
vow_config	qca_ol.ko
max_descs	qca_ol.ko
qwrap_enable	qca_ol.ko
max_peers	qca_ol.ko
max_vdevs	qca_ol.ko
sa_validate_sw	qca_ol.ko
dfs_disable	qca_ol.ko
frac	qca_ol.ko
intval	qca_ol.ko
ar900b_20_targ_clk	qca_ol.ko
qca9888_20_targ_clk	qca_ol.ko
otp_mod_param	qca_ol.ko
cfg_iphdr_pad	qca_ol.ko
emu_type	qca_ol.ko
enable_smart_antenna	qca_ol.ko
max_active_peers	qca_ol.ko
low_mem_system	qca_ol.ko
nss_wifi_olcfg	qca_ol.ko
nss_wifi_ol_skip_nw_process	qca_ol.ko
ol_scan_chanlist	qca_ol.ko
fw_code_sign	qca_ol.ko
testmode	qca_ol.ko
lteu_support	qca_ol.ko
bmi	qca_ol.ko
wari	qca_ol.ko
war1_allow_sleep	qca_ol.ko
allocram_track_max	qca_ol.ko
max_clients	qca_ol.ko
max_vaps	qca_ol.ko
fw_dump_options	qca_ol.ko

<b>module_param</b>	<b>module_name</b>
enable_mesh_support	qca_ol.ko
wmi_ring_size	qca_ol.ko
ahbskip	umac.ko
enable_mesh_peer_cap_update	umac.ko
wifiposenable	umac.ko
atf_mode	umac.ko
atf_msdu_desc	umac.ko
atf_peers	umac.ko
atf_max_vdevs	umac.ko
enable_pktlog_support	umac.ko

## Module insertion Order

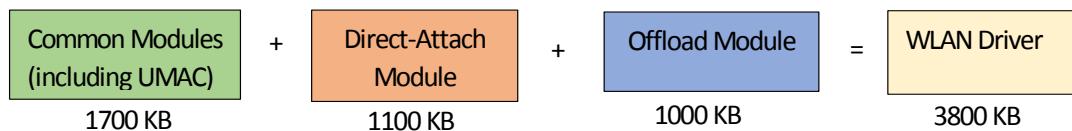
The WLAN Driver modules need to be inserted in the following order, due to their dependencies.

```
ASF
ADF
ATH_DFS
ATH_SPECTRAL
UMAC
ATH_HAL
ATH_RATE_AHTEROS
HST_TX99
ATH_DEV
QCA_DA
QCA_OL (this module can be inserted after UMAC)
```

The qca\_ol will bring up OL devices and qca\_da will bring up DA devices. As they are independent of each other, depending on which module is inserted first, that device will come up first. This will decide the radio name as well.

## WLAN Driver sizes

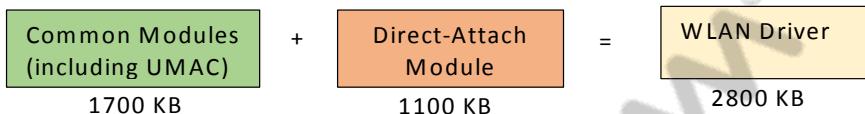
The following figure illustrates the driver sizes for platforms using both OL and DA chipsets:



The following figure illustrates the driver sizes for platforms using only OL chipsets:



The following figure illustrates the driver sizes for platforms using only DA chipsets:



**Figure 1-12 WLAN Driver Sizes**

## 1.6 Wi-Fi Calibration Data mapping

To enable the WLAN driver to obtain the correct calibration data, the following entities must be in synchronization:

- Calibration data stored in flash.
- Pre-initialization scripts
- Wi-Fi scripts

### 1.6.1 Calibration data stored in flash

For platforms with multiple radios, the calibration data is stored in the flash in a particular order. This order is determined when the board is calibrated.

### 1.6.2 Pre-initialization scripts

These scripts read the calibration data from flash and write to file. The data is normally written to the "tmp" directory. The files are named with the radio names such as wifi0.caldata, wifi1.caldata. These scripts must have knowledge of both the calibration data in the flash and the order in which the WLAN driver brings up the radios. These scripts must write the filenames accordingly so that when the WLAN driver comes up, the correct calibration data is used.

The pre-initialization scripts are available at the following locations:

For IPQ platforms: qsdk/target/linux/ipq806x/base-files/lib/read\_caldata\_to\_fs.sh

For QCA9531/QCA9558/QCA9563 platforms: qsdk/target/linux/ar71xx/base-files/lib/preinit/81\_load\_wifi\_board\_bin

These scripts already have access to the type of the board by reading the /tmp/sysinfo/board\_name. Using the board name, decisions are made to write the calibration data filenames in a particular order.

### 1.6.3 Wi-Fi scripts

The Wi-Fi scripts insert the WLAN modules, which in turn bring up the radios. If there are multiple radios (Offload and Direct-attach), the order of inserting the modules determines the order in which the radios are to be brought up, and also determines the radio names (such as wifi0 and wifi1)

The qca.ol.ko initializes the offload radios. The qca.da.ko initializes the direct-attach radios. As described earlier in this section, the pre-initialization scripts contain knowledge about the order in which WLAN modules are inserted and depending on that order, the calibration data filenames are chosen.

QUALCOMM  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

# 2 WLAN AP Driver Layers

This chapter describes the WLAN AP driver layers and the corresponding data structures or objects for these layers. It also describes the code base structure and provides information on directory and common data structure, layers, and source tree layout.

## 2.1 Major data structures

The entire WLAN driver processing and driver access are done through the below data structures. Each layer has its own set of data structures and any access to these layers happens through their corresponding data structures reference and APIs provided by that layer. No global data are used. This enables WLAN driver to support various AP platform configurations like multiple radios, multiple BSS on same radio device, and so on.

Figure 2-1 shows the major objects in all WLAN driver layers and how they are linked with each other. The detail description of these objects are described in the following sections.

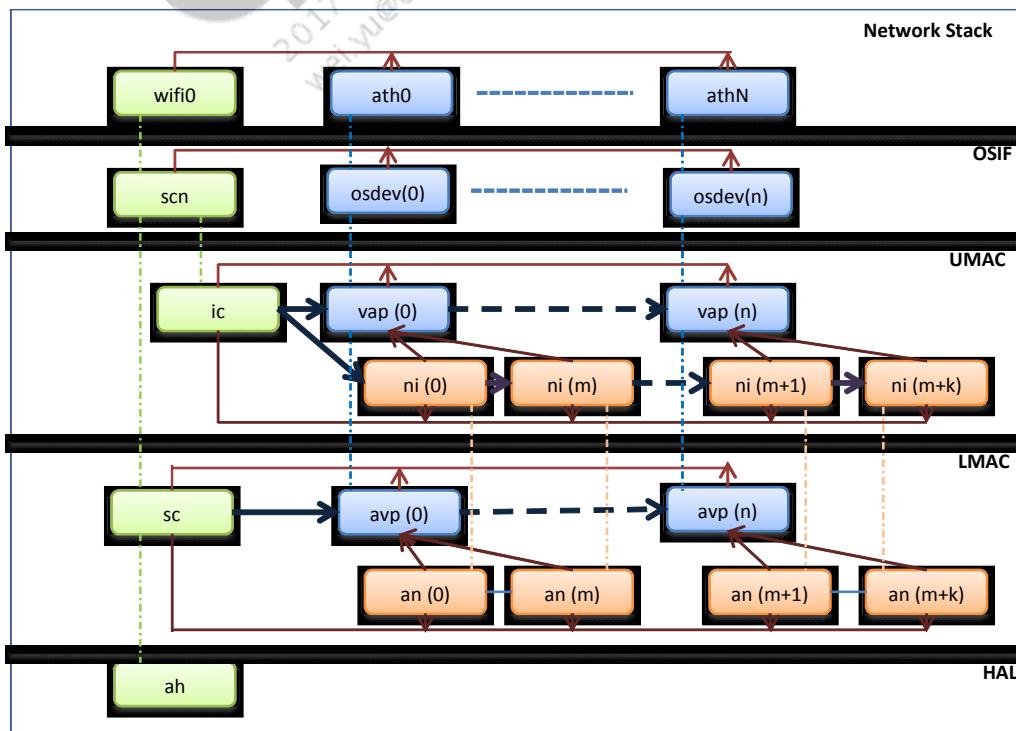


Figure 2-1 WLAN Driver Major Objects

## 2.1.1 OSIF data structures

### 2.1.1.1 ath\_softc\_net80211

ath\_softc\_net80211 is an OS interface radio device structure. It is typically referred to as “scn” inside the OSIF layer.

This is the OS interface abstraction provided by the driver for each WLAN radio interface in the AP platform. This structure maps to the network device private structure provided by the networking stack. This object maps to the “wifi” interface on the network stack. The following are some of the data stored in this structure.

- UMAC common device structure
- Reference to LMAC device handle
- LMAC interface functions
- OS device handle
- Locks for synchronization
- Reference to node to key index mapping.

### 2.1.1.2 osif\_dev

osif\_dev is an OS interface device structure. It is defined as type os\_if\_t and typically referred to as “osdev” inside the OSIF layer.

This is OS interface abstraction provided by the driver for each WLAN network interfaces created in the system. This structure maps to the network device private structure provided by the OS stack. This helps to maintain driver-specific information for this network device in the OS interface layer. This object maps to the “ath” interface on the network stack. Some of the data maintained in this structure are

- Reference to network device
- Reference to parent network device
- Opaque pointer to UMAC wlan interface – used in UMAC API calls
- Opaque pointer to UMAC common interface (radio interface) – used in UMAC API calls
- VlanID and vlgrp
- Handle to UMAC connection state machine and scan related handlers

## 2.1.2 UMAC data structures

### 2.1.2.1 ieee80211com

ieee80211com is a UMAC radio device structure. It is defined as type wlan\_dev\_t and typically referred to as “ic” inside the UMAC.

The UMAC abstracts all the radio-specific information into the ieee80211com structure. These data are common to all the network interfaces which share the common radio interface. The state shared by the underlying device and the UMAC layer is exposed here.

Some of the information stored in this object are

- Opaque handle to osdev
- Reference to scan table and scanner object
- List of all UMAC network interface object
- List of all UMAC Node object (pointer to Node table) for connection management.
- Device specific parameters which are common to all VAPS and nodes
- Function pointers (APIs) to access the device specific parameters and data

### 2.1.2.2 ieee80211vap

ieee80211vap is a UMAC network interface structure. Each WLAN networks interface is represented as a “Virtual AP” (VAP) in the UMAC and shares a single underlying physical device. Each VAP represents the operating mode of the VAP which could be a host Access Point, IBSS or Station modes. Each VAP has a corresponding OS device entity through which traffic flows and for which the applications use to issue ioctls, and so on. Information stored in this object includes

- Opaque handle to osdev
- BSS node reference
- All VAP specific data structures.

### 2.1.2.3 ieee80211\_node

UMAC Node structure. Defined as type wlan\_node\_t and typically referred to as “ni” inside UMAC

A node could represent a BSS in infrastructure network, an ad-hoc station in IBSS mode, or an associated station in HOSTAP mode. The set of ieee80211\_node objects reveals the device’s local view of the network. For example, for a device operating in station mode, the only ieee80211\_node object is the AP with which the station associates. For a device operating in AP mode, the set of ieee80211\_node objects represents the set of stations in the current BSS. For a device operating in ad-hoc mode, the set of ieee80211\_node objects represents the visible neighbors (within the IBSS or not).

## 2.1.3 LMAC data structures

### 2.1.3.1 ath\_dev: LMAC device structure

Defined as ath\_softc and as type ath\_dev\_t (opaque reference to other layers access). Typically referenced as “sc” inside the LMAC code.

The LMAC encapsulates the low-level driver components into a Qualcomm Technologies device object. The ath\_dev is an opaque object to upper layers and can be accessed only through a well-defined function table (ath\_ops) referred to as LMAC APIs. The advantage is that regardless of the underlying hardware, the protocol stack can treat the device as opaque and access the device object in the exact same manner, which is through the function table. For example, a protocol stack can access Qualcomm Technologies PCI-based devices and USB-based devices the same way, without worrying about the internal difference of the two product families.

### 2.1.3.2 ath\_vap: LMAC vap structure

Typically referenced as “avp” inside LMAC code.

Each ath\_vap object represents a VAP (virtual AP) entity. This object directly maps to the ieee80211vap object in the UMAC. By default, each physical device has only one VAP entity, and additional VAPs can be added or removed by protocol stack. Each ath\_vap object can be accessed by upper layers through a unique index. Qualcomm Technologies hardware can support up to 16 VAPs operating simultaneously, so the valid index is from 0 to 15.

### 2.1.3.3 ath\_node: LMAC node structure (corresponds to each UMAC node)

Defined as type ath\_node\_t and typically referred as “an” in the code base.

Each ath\_node object represents its corresponding ieee80211\_node object and stores LMAC specific information for that node. Similar to ath\_dev, each ath\_node is exposed to the upper layer as an opaque object and accessed through the ath\_ops table.

## 2.1.4 HAL data structures

### 2.1.4.1 ath\_hal: HAL device structure

Typically referenced as “ah” in the code.

Clients of the HAL call ath\_hal\_attach to obtain a reference to the ath\_hal structure for use within the device. Hardware-related operations that follow must call back into the HAL through interface, supplying the reference as the first parameter. This structure abstracts the underlying hardware chip data structures to the upper layer and provides a common interface. The actual device specific HAL structures are defined separately for each hardware chip family (for example: ath\_hal\_9300, ath\_hal\_5416, and ath\_hal\_5212).

## 2.2 Directory structure

The following are the most relevant directories:

```
drivers/wlan  
  /os  
  /linux  
  /win_nwf/  
  /darwin  
  ...  
  /hal  
  /ar9300  
  ...  
  /lmac  
  /ath_dev  
  /ratectrl  
  /umac  
  /mlme  
  /txrx  
  /crypto  
  /qca_da
```

## 2.3 Common data structures

### 2.3.1 vap

- Abstraction for an interface
- Can be multiple VAPs over a single physical interface

### 2.3.2 node

- Per-connection structure
- Each layer owns and maintains relevant part of the node

### 2.3.3 ic

- Device abstraction at the *net* and higher layers

### 2.3.4 sc

- Device at the *ath* layer

### 2.3.5 ath\_hal

- HAL data structure

## 2.4 Driver layers

The host driver architecture is comprised of the OS shim, 802.11 protocol, device, and HAL layers. The following sections provide more details on these layers.

### 2.4.1 OS shim layer

The OS shim layer provides the following functions:

- Driver initialization and loading
- Interrupt hook-up
- interface sending

There is one OS shim layer per operating system. Examples are Native Wifi, NetBSD, Darwin, Linux, eCos, and VxWorks. The OS shim layer interfaces OS functions to common code, and contains:

- OS specific initialization code (such as interrupt/packet buffer allocation)
- library of common functions (such as locking/malloc)
- OS entry points for transmit and interrupt (including receiving)

#### 2.4.1.1 OS shim functions

The OS shim layer functions are contained in **osdep.h** (wlan/os/darwin/include) and **ath\_osdep.c** (wlan/os/darwin/src), and have OSIF configuration and functions such as

- OS\_SLEEP
- OS\_GET\_TIMESTAMP
- OS\_TIMER
- spin\_lock\_init
- spin\_lock
- spin\_unlock
- Struct ath\_reg\_params ath\_params = {};
- ath\_driver\_command – ioctl interface
- ath\_dev\_start
- ath\_dev\_stop

### 2.4.2 802.11 protocol layer (net, UMAC)

The 802.11 protocol layer (also known as the net or UMAC layer) contains the complete control path for connection establishment. It is implemented as hierarchical state machines (such as connectionSM → AssocSM). The UMAC provides packet encapsulation and decapsulation facilities (dot3 ↔ do211). The UMAC layer handles resource management, where multiple VAPs

share a common channel and hardware. Additionally, the UMAC layer also provides the following functions:

- Power Management
- MLME
- Scanning
- Encryption

#### 2.4.2.1 UMAC-to-LMAC interface

UMAC to LMAC calls go through a switch table. The table is passed to the UMAC during a **ath\_attach()** call. The code is located in **lmac/ath\_dev/ath\_main.c**:

```
struct ath_ops {} ath_ar_ops = {};
```

Sample calls:

```
ath_vap_attach
ath_vap_detach
ath_tx_init
ath_rx_init
ath_tx_start_dma
ath_set_channel
ath_get_channel
ath_intr <isr>
ath_handle_intr <dpc>
```

#### 2.4.3 Device layer (ath, LMAC)

The device layer provides the following functions:

- Transmit
- Receive
- Interface between network and HAL (“pass-through” interface functions)
- Tx aggregation
- Rx reordering
- DMA
- Interrupt handling

#### 2.4.3.1 LMAC-to-UMAC interface

LMAC-to-UMAC calls go through a switch table, similar to a UMAC-to-LMAC call. The table is passed to the UMAC using **ath\_dev\_attach()** during an **ath\_attach()** call. The code is located in **umac/if\_lmac/if\_ath.c**:

```
struct ieee80211_ops net80211_ops = {};
```

Sample calls:

```
ath_net80211_tx_complete
ath_net80211_rx
ath_net80211_input
ath_net80211_set_countrycode
```

### 2.4.3.2 LMAC-to-HAL interface

The LMAC attaches to the HAL using **ath\_dev\_attach()** during an **ath\_hal\_attach** call. The code is located in **wlan/hal/ah.c**. **ath\_hal\_attach** uses the passed-in deviceid to attach the specific device functions.

For example, **ar9300Attach()** is called for the AR93xx chips. The code is located in **wlan/hal/ar9300/ar9300\_attach.c**. This returns a call table specific to the AR93xx chips and includes functions such as

```
ar9300EnableReceive
ar9300StartTxDma
ar9300SetTxDp
ar9300FillTxDesc
ar9300ProcRxDesc
```

### 2.4.4 HAL

The **Hardware Abstraction Layer** contains a library of functions to access the hardware.

The HAL implements functions specific to one particular chip family:

- Low-level chip initialization
- Per-chip radio configuration (through an INI file; can be platform-specific if required)
- PHY control
- Register-level interface to hardware
- Tx/Rx descriptors handling
- Tx/Rx data structure formation

This part of the code is meant to be as OS independent as possible to support a wide variety of platforms. The HAL layer is mainly called by the LMAC, in a way similar to a library. The files for the HAL layer are located in the **hal** directory.

For this reason, the HAL is built separately from the rest of the driver. It uses dedicated makefiles, and the result of this makefile is then used in the driver to be linked with the other driver sub-parts.

The following sub-directories are present into this driver part:

#### 2.4.4.1 hal/ar5212

This directory contains the source code to manage the 11g atheros chips family. It implements chip dependent functions for the following chipsets:

- AR23xx
- AR24xx

- AR52xx
- AR5413

#### 2.4.4.2 hal/ar5416

This directory contains the source code to manage the first 11n WLAN generation architecture chipsets. It contains the source code functions of the following families:

- AR5416
- AR91xx
- AR92xx

#### 2.4.4.3 hal/ar9300

This directory contains the source code to manage the last 11n WLAN generation chipsets. The following families are included in this directory:

- AR93xx
- AR95xx

#### 2.4.4.4 hal/linux

This directory contains the Linux specific parts of the HAL. It also includes the Makefiles to build the HAL on multiple Linux based SoC (in the public directory).

#### 2.4.4.5 hal/nartlinux

This directory is used mainly for the NART production software. It contains some headers specific to that application.

### 2.5 Source tree layout

The QCA WLAN driver is divided into multiple directories. The following are brief descriptions of the content of this architecture.

#### 2.5.1 asf

The **Atheros Service Framework** provides some services to the WLAN driver in an OS-independent manner. It contains some functions to handle print and memory management; these functions are then called from the driver to make the core driver OS independent.

#### 2.5.2 qdf

The **Qualcomm Driver Framework** provides a mechanism to run the Qualcomm WLAN driver on a variety of operating systems and platforms. In a sense, it's close to the ASF layer, but is

handling a different function set. Network stack and network buffer management are abstracted by this layer.

### 2.5.3 htc

For full offload architecture, the **Host Target Communication** layer implements the communication protocol used between the host and the target. It contains the code to encapsulate the packets into the QCA-defined header for configuration and data traffic.

These functions rely on a defined sub-layer API, and abstract the hardware bus used to communicate with the device. It supports xMII, PCI(e) and USB. The hardware bus abstraction function is defined in the “htc/hif” directory.

### 2.5.4 include

The include directory contains the main headers used in the WLAN driver. Most of the headers are located there, however some other headers are located directly into the other directories with their corresponding source code.

### 2.5.5 lmac

The **LMAC** (Lower Media Access Controller) is the lower part of the WLAN driver. Functions defined in this directory are used in the datapath. The LMAC takes frames to and from the UMAC, and configures the datapath to transform the frame into a format that the hardware will understand.

The LMAC code calls the HAL for any chip-specific access. The directories named in the following subsections are defined in the LMAC:

#### 2.5.5.1 lmac/ath\_dev

The **ath\_dev** directory is the LMAC core. It contains all the code to process the frame between the upper layer interface (UMAC) and the chipset hardware. This represent a large portion of the datapath.

#### 2.5.5.2 lmac/ath\_pktlog

The **ath\_pktlog** directory contains the code that implements the packet log feature. It can store packets of information into a ring buffer, and this ring buffer can then be accessed using “/proc”, and analyzed using Perl scripts.

#### 2.5.5.3 lmac/dfs

The **dfs** directory implements the DFS logic into the LMAC.

#### 2.5.5.4 lmac/ratectrl

The **ratectrl** directory implements the rate control logic of the driver. This code is called by the code in **ath\_dev** to determine the best rate to use for a specific packet.

#### 2.5.5.5 lmac/spectral (only while using the SPECTRAL package)

The **spectral** directory implements the spectral scan feature. It is compiled only when using the right driver option to enable the feature.

#### 2.5.5.6 os

The **os** directory contains multiple sub directories to build/use/interface the driver with the operating system.

#### 2.5.5.7 os/linux

The Linux directory contains all the code to interface most of the driver (expect the HAL) to Linux. It contains the Makefiles used to build the driver, header files, a Linux-specific interface API (such as iwconfig, iwpriv), and commands.

The main driver Makefile os/linux/**Makefile** contains the targets to recursively parse the driver directories and build the necessary files.

Also, the os/linux/**tools** directory contains some CLI commands used to setup and configure the driver on the gateway.

### 2.5.6 umac

The **UMAC** (Upper Media Access Controller) implements the 802.11 protocol layer of the WLAN driver. Its role is to be the interface between the operating system and the LMAC. The following directories are part of the UMAC.

#### 2.5.6.1 umac/acl

The ACL (Access Control List) directory contains the code implementing the ACL feature. When a station connects to the AP, the ACL decides to allow or reject the STA based on the ACL policy and configuration. The secondary ACL is also implemented that is used for band-steering purpose. Both lists share the same structure and are differentiated using flags.

**NOTE** Secondary ACL is meant for band-steering purposes only. Ignore ACL feature does not work with this list.

**NOTE** Secondary ACL list is used for blacklisting and the primary ACL is used for whitelisting purpose. Both should be used exclusively.

### 2.5.6.2 umac/acs

The **acs** (Automatic Channel Selection) directory contains the algorithm to automatically select the channel at WLAN start. It parses the authorized channels and selects which one is the most appropriate.

### 2.5.6.3 umac/base

The **base** directory contains some general purpose functions used within the driver to access such items as node information, channel, VAPs. It is used in many locations within the UMAC.

### 2.5.6.4 umac/crypto

The **crypto** directory contains the code to handle cryptographic related structs. It is called by the UMAC datapath functions to encapsulate/decapsulate packet depending on their encryption, and it is also called at configuration stage to set the driver encryption mode.

### 2.5.6.5 umac/extap

This directory implements the **Extension AP** feature. It is used on the AP to connect additional devices to the STA.

### 2.5.6.6 umac/if\_lmac

This layer is in charge of making the transition between UMAC and LMAC. It implements the functions called by the LMAC from the UMAC and from the UMAC to the LMAC.

### 2.5.6.7 umac/include

The include directory contains multiple headers defining functions and structure used in the UMAC.

### 2.5.6.8 umac/ique

**Improved Quality User Enhancement** implements two features used specifically for video performances.

HBR (Headline Block Removal) blocks the outgoing UDP frames to a node if the connection to this node is unreliable to avoid impacting other nodes.

ME (Multicast Enhancement) converts one multicast streams into multiple unicast streams to enhance video performance.

### 2.5.6.9 umac/mlme

The **MAC Layer Management Entity** directory implements the layer with the same name in the 802.11 specification. It generates/receives the management packets, and triggering all the features related to these packets: time synchronization, scanning, probes, and so on.

### 2.5.6.10 umac/p2p

The **Peer-to-Peer** directory implements the WiFi Direct protocol as described in WFA.

### 2.5.6.11 umac/pm

The **Power Management** directory implements the power save algorithms for AP and STA.

### 2.5.6.12 umac/regdmn

The **Regulatory Domain** directory implements all the region management code to set the available channel list corresponding to the EEPROM and user configuration. It also contains the code to populate the corresponding IE.

### 2.5.6.13 umac/resmgr

The **Resource Manager** directory contains some functions to handle the driver global resources. These functions are called from multiple locations within the driver to allocate and free these resources.

### 2.5.6.14 umac/rpt\_placement

The **Repeater Placement** directory implements the repeater placement feature. It includes code to initiate, encapsulate and decapsulate the packets exchanged during the placement phase. It also includes the functions to configure this feature through the CLI.

### 2.5.6.15 umac/scan

The **scanning** directory contains the AP and STA scan state machine. This process is initiated by a request from the MLME, but the code containing this algorithm is in this directory.

### 2.5.6.16 umac/smart\_antenna

The **Smart Antenna** directory contains the state machine to handle the smart antenna feature. It has code to configure it, and it also implement functions that are called from within the datapath to decide which smart antenna configuration to use (in Tx) and process the smart antenna-specific packets.

### 2.5.6.17 umac/sme

The **State Machine Entity** is used in case the STA needs to run the authentication and association process. It contains a state machine that will update the current state depending on the frames received from the AP.

### 2.5.6.18 umac/txbf

The **Transmit Beamforming** directory contains the code used to trigger and send the transmission of a CV update request. The triggering of these CV update is based on timers.

### 2.5.6.19 umac/txrx

The **Transmit-Receive** directory contains the main datapath of the UMAC. One file (ieee80211\_output.c) handles the Tx datapath, and another file (ieee80211\_input.c) handles the Rx datapath. These functions are the core functions of the UMAC as they are in charge of transferring the packets between the OS interface and the LMAC. Most of these UMAC features described in this section are called from this location.

### 2.5.6.20 umac/vi\_dbg

The **Video Debug** directory implements a tool to debug specifically the video features. It has the capability to detect some user-defined packets and report information in packet log.

### 2.5.6.21 umac/wds

The **Wireless Distribution System** directory implements the WDS feature as described in the 802.11 specification. It implements the code to handle WDS connection/disconnections, and encapsulate/decapsulate the WDS frames. It has code to support both standard WDS and NA-WDS (Non Associated WDS).

### 2.5.6.22 wow

The **Wake on Wireless** directory contains the code to handle this feature. It includes chip programming to enter WoW mode, and WoW frames parsing. This directory is building a separate module.

## 2.6 Improvements to WMI layer architecture

This section describes the enhancements implemented for the WMI layer in WLAN platforms to provide a framework for the support of TLV (for future chipsets) and non-TLV format. This functionality caters to the following requirements:

- Rearchitecture of wireless infrastructure networking (WIN) WMI layer
- Support for WMI TLV-based implementation
- Support for WMI non-TLV-based implementation
- Support APIs to aid different context switching requirements of upper layers

The following architectural enhancements are implemented:

- WMI API is unified across WIN designs irrespective of non-TLV or TLV based target. No API changes will be made in future releases after WMI layer rearchitecture.

- Support for Receive path ramifications to aid upper layer to use context as per its design. WMI Receive path does not assume any context independently. The WMI Receive path executes in the context of the callbacks provided.
- WMI does not assume any serialization nor enforce any serialization.

**NOTE** Firmware might require WMI commands to ingress in sequence and Out of order WMI command might not be expected

WMI must assume that this must be serialized by upper layers. This condition also enables WMI to have stateless design and does not need to maintain context.

- WMI must support multiple radios for WIN. The API is unified across WIN and upper layer needs to ensure proper radio id is filled before passing to WMI.
- WMI must support multiple System-on-Chips (SoCs). It must be possible to instantiate WMI per SOC.
- WMI must depend on modules for OS and debug support.

## 2.6.1 WMI layer design

The following figure illustrates the high-level interaction of WMI layer:

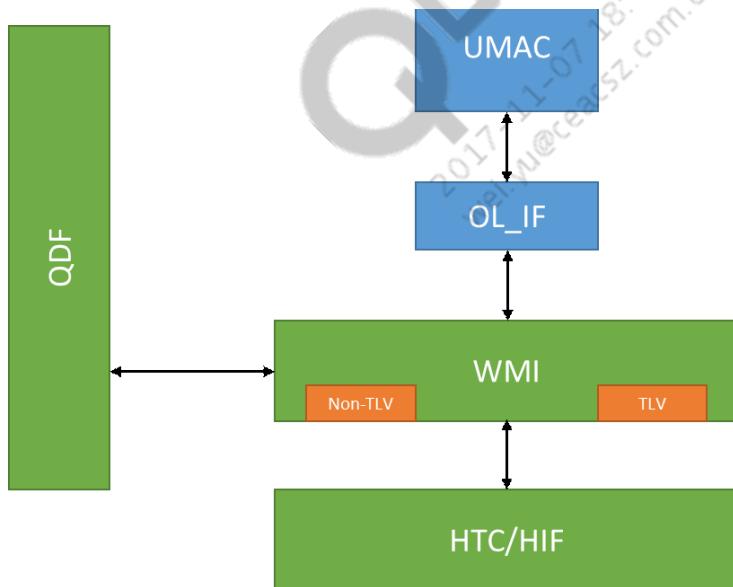


Figure 2-2 WIN WMI design

## 2.6.2 WMI layer interactions with other modules

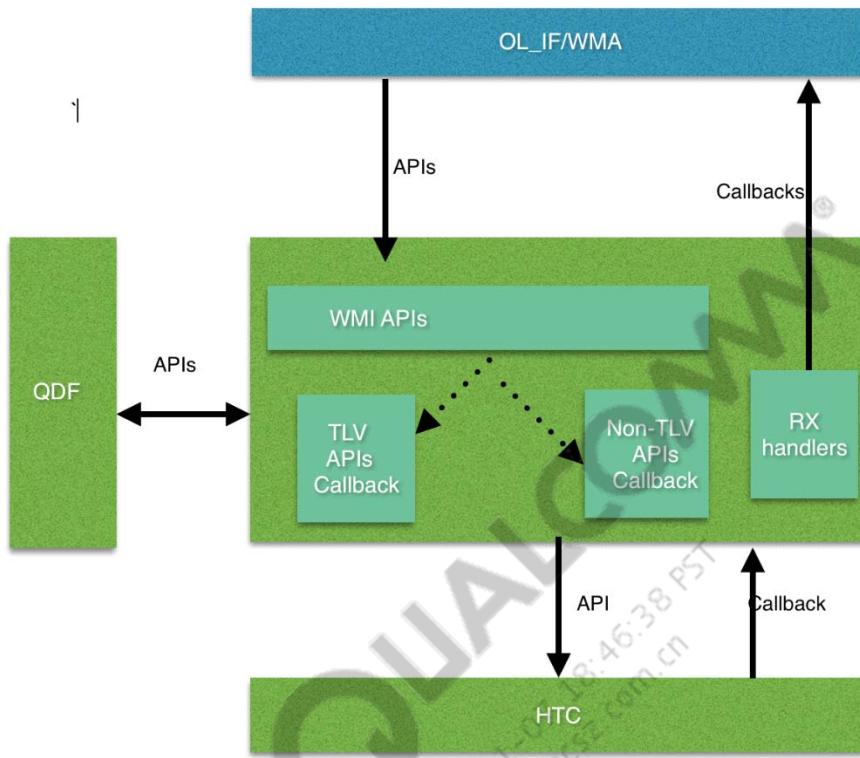


Figure 2-3 WMI interactions

- Northbound is with OL\_IF (WMI exports APIs and OL\_IF registers callbacks).
- Southbound is with HTC (APIs exported by HTC and callback registered by WMI).
- WMI interacts with QDF for all OS related services such as memory allocation and locks (APIs exported by QDF).

WMI layer is common for both non-TLV and TLV based WIN targets. The goal is to maintain common design to support both message formats. This design must provide a consistent interface to the WIN upper layer irrespective of the underlying target. To satisfy this requirement, WMI exports common set of APIs and data structures that can be used to send WMI messages and extract data from received WMI message.

WMI layer exports APIs and data structures through header files to be used by upper layers. This file has definitions of all data structures to be used by upper layers while talking to WMI. Upper layers can use these APIs to form and transmit a WMI message.

Abstraction of TLV vs non-TLV processing is hidden within WMI layer. WMI layer will have implementations for both TLV and non-TLV WMI messages. Since `wmi_unified.h` is controlled by FW, WMI will have two versions of `wmi_unified.h` (one for TLV and one for non-TLV). TLV and non-TLV functions will be implemented separately. During WMI initialization, appropriate handlers are set, based on the target type. TLV and non-TLV implementation are separate and include appropriate header files from FW. This implementation ensures that no conflicts occur

between TLV and non-TLV WMI enums and data structures. Also, this design enables non-TLV implementation to be easily excluded, if it is not required by upper layers (for example, only TLV based targets).

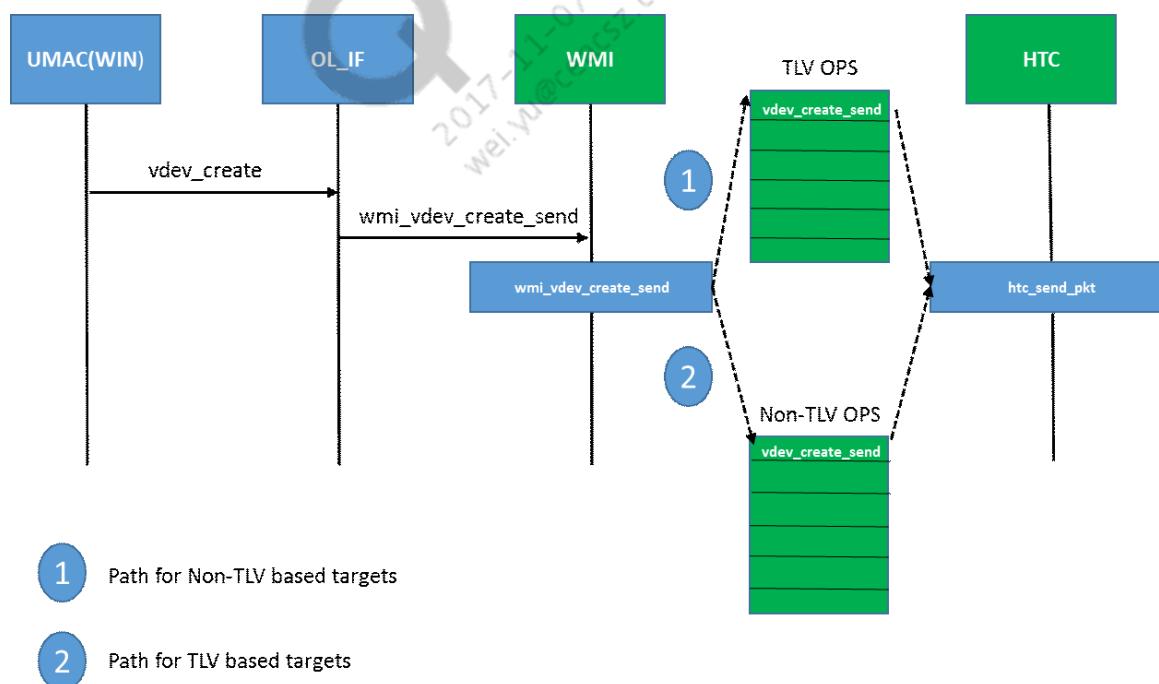
## WMI transmission

WMI uses TLV format and WMI command ID to transmit WMI commands to firmware. TLV commands are constructed, based on the interface (`wmi_unified.h`) shared across the host and firmware.

WMI has defined `wmi_unified_cmd_send` API for transmit which include WMI handle, WMI buffer (TLV buffer) and WMI command ID. WMI handle is handle of the WMI instance, WMI buffer contains TLV command buffer for each command ID.

WMI Transmit is asynchronous in nature and there is no response associated in return. In case there is response associated with a command, they are handled as event from firmware. Because commands are asynchronous, WMI does not hold any blocking context. This is responsibility of the upper layers to have a blocking context whenever the upper layer wants to handle the response in same context.

WMI transmit path handles TLV and non-TLV implementation based on the target type. During WMI attach, corresponding TLV and non-TLV callbacks are attached based on target type provided by the UMAC. TLV and non-TLV callbacks are abstracted in WMI layer hidden to the UMAC, that has unified WMI interfaces across TLV and non-TLV-based targets.



**Figure 2-4 WMI Transmit showing vdev create using WMI command WMI\_VDEV\_CREATE\_CMDID**

## WMI receipt

In Rx, the Rx event registration mechanism that existed until QCA\_Networking\_2016\_SPF.3.0 remains unchanged. WMI must provide APIs to extract data from received WMI message. RX handling functions can call appropriate APIs to extract data without worrying about TLV vs non-TLV formatting. This API model provides following flexibility to upper layers

- Rx event registration mechanism can be made simpler, since it need not account for different data structures in WMI messages.
- If upper layer does not have to support both TLV and non-TLV, it can directly type-cast and use data structures.

WMI supports RX callback in following execution context -

- Default Tasklet context
- Worker context

RX registration API will provide an argument to choose the context in which Upper layer wants this event to be handled. Based on the execution context requested, RX event callback will be called in that context.

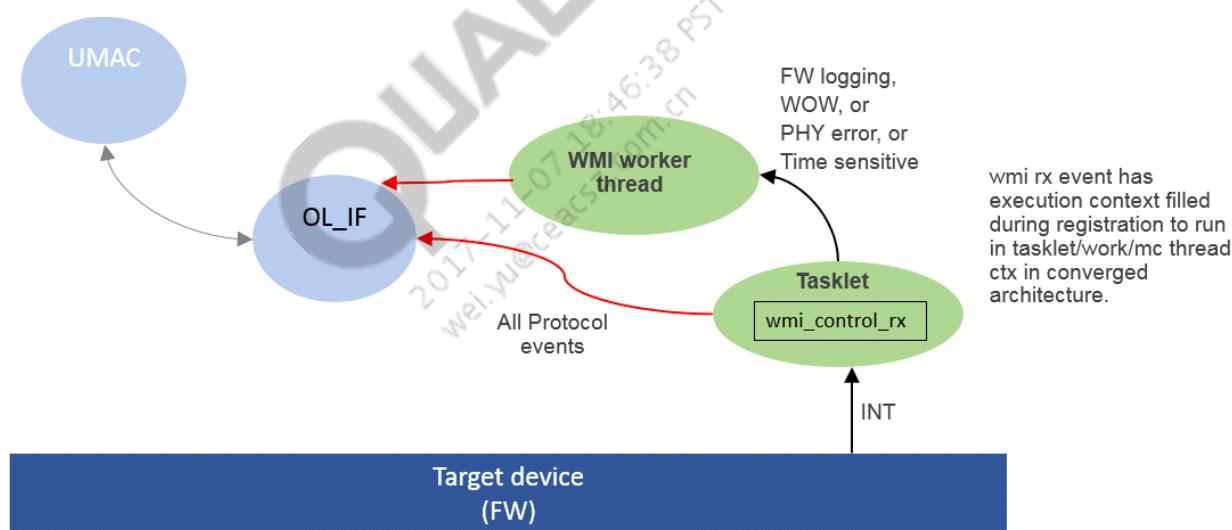


Figure 2-5 WMI rx path for WIN design

### 2.6.3 WMI initialization sequence

WMI initialization happens similar to the mechanism that existed before the implementation of this feature of WMI layer enhancements:

1. `wmi_unified_attach` is called during device probe. Creates WMI instance and instantiate WMI APIs to either TLV or non-TLV based APIs.
2. `wmi_unified_connect_htc_service` - After attach, WMI needs to connect to HTC to get the endpoint id. This endpoint ID is used for further interaction with HTC to send and receive WMI messages.

3. Firmware sends WMI\_SERVICE\_READY indicating that it is ready and provides its capabilities.
4. Based on supported services, host sends WMI\_INIT to firmware with target config.
5. Firmware sends WMI\_READY event to indicate it is ready.
6. Depending on supported service there can be a WMI\_EX\_SERVICE\_READY event message before host sends WMI\_INIT.

After receiving WMI\_READY, target is ready and can accept further WMI messages and send events. WMI provides APIs to store supported service bitmap and to query supported services.

#### 2.6.4 WMI backward compatibility

TLV based implementation Radio id is backward compatible. There available in all PDEV commands, which is a reserved field in messages to include pdev\_id.

Any new additions adhere to the TLV rules and guidelines, which ensure backward compatibility. If there is any message construction that needs to fill Radio Id field in certain fields based on the FW capability or version, WMI takes care of filling the message accordingly. If there is any change in the sequence of message due to FW implementation, upper layer ensures backward compatibility by following the correct sequence based in FW capability/version.

#### 2.6.5 Cookie or pool-based allocation for HTC packets

In the new WMI design, cookie/pool based allocation for HTC packet is removed. Allocation is performed from OS memory allocation APIs. Atomic counters are maintained to keep track of outstanding WMI messages.

#### 2.6.6 WMI directory structure

The following illustration displays the hierarchy of the directories:

```
wmi-----
  |-inc-
  |   |- wmi_unified_priv.h
  |   |- wmi_unified_api.h
  |   |- wmi_unified_param.h
  |-src-
    |- wmi_unified.c
    |- wmi_unified_tlv.c
    |- wmi_unified_api.c
    |- wmi_unified_non_non_tlv.c
    |- wmi_tlv_helper.c
    |- wmi_tlv_platform.c
```

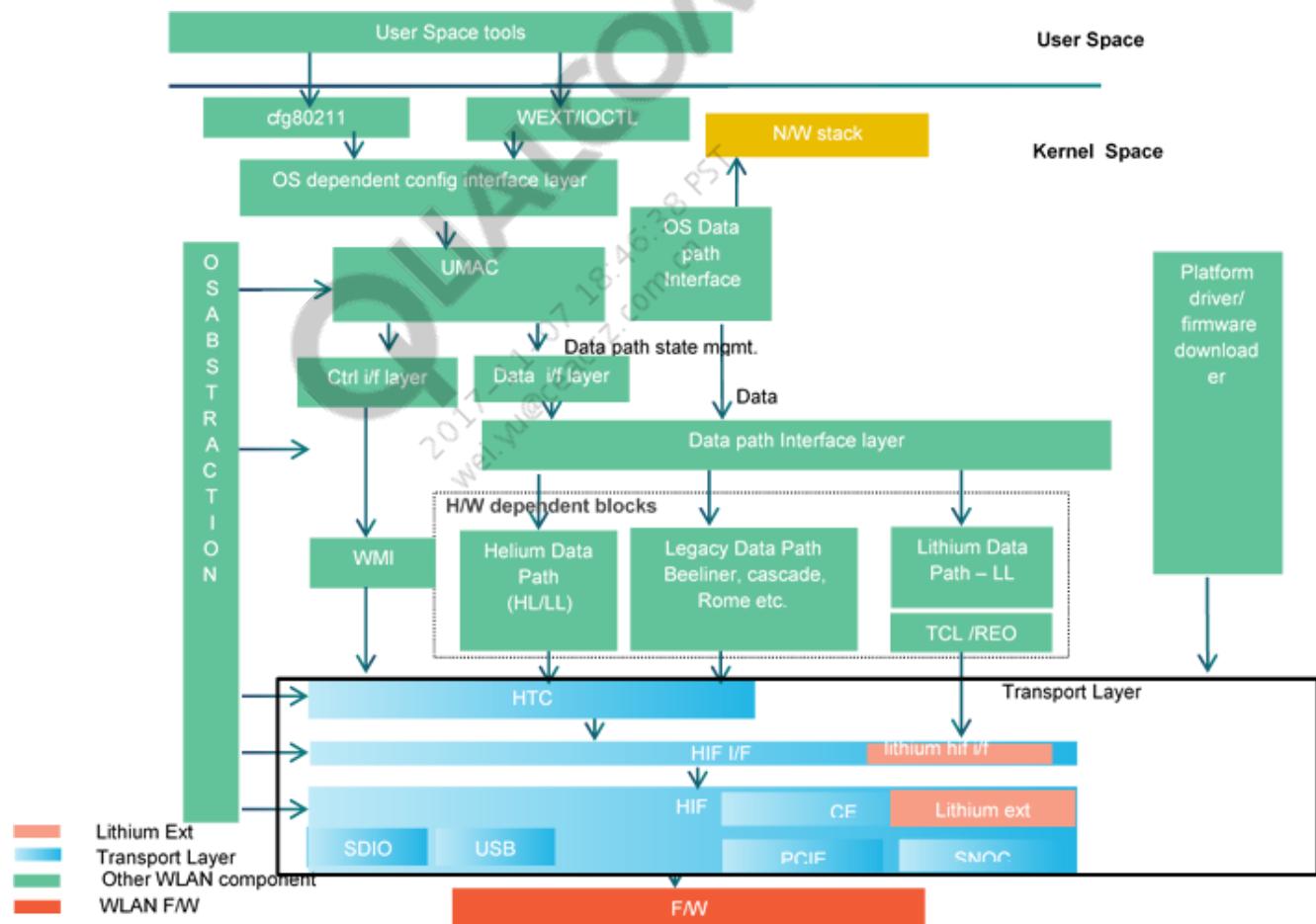
The files in wmi/inc provide API and data structures abstracted to upper layer by WMI. Upper layers only use APIs and data structures exposed in these header files. Also, the files in wmi/src implement the APIs exposed to upper layer. All the TLV and non-TLV message construction are implemented in these files.

## 2.7 Transport layer enhancements for modularization

This section describes the enhancements done on transport layer (host-target communication [HTC] and host-interface node [HIF]). Cleanup and enhancements of transport layer code are performed for better modularization. The following are the benefits of this feature:

- Cleaner API for interaction with upper layer.
- Prevention of access of internal data structure of HIF by upper layer and vice-versa.
- Concurrent support for multiple interface such as PCI USB/SDIO.
- Infrastructure to enhance the transport layer for future-generation chips.

### 2.7.1 Host architecture



The Host-Firmware interaction is not modified. The public API of HTC/HIF is changed. As a result, all the WLAN modules adopt the new APIs.

## 2.7.2 Host Low Level Design

This subsection describes the design enhancements made to the host component for the transport layer:

### 2.7.2.1 Changes to Public Interfaces

The earlier upper layer code used access the HIF device structure update or use information as part of clean up all the access to the HIF internal structure has been removed and a cleaner API interface has been provided for upper layer to access the hif.

The interaction to HIF for the upper layer is through an opaque handler `hif_opaque_softc`

### 2.7.2.2 Initialization/Shutdown

New APIs are added for HIF initialization and shutdown . All the bus related information has been moved to HIF instead of handling these in the Upper layer

The following are the new APIs

1. `hif_open`
2. `hif_enable`
3. `hif_disable`

#### `hif_open`

```
/***
 * hif_open(): hif_open
 * @qdf_ctx: QDF Context
 * @mode: Driver Mode
 * @bus_type: Bus Type
 * @cbk: CDS Callbacks
 *
 * API to open HIF Context
 *
 * Return: HIF Opaque Pointer
 */
struct hif_opaque_softc *hif_open(qdf_device_t qdf_ctx, uint32_t mode,
                                    enum qdf_bus_type bus_type,
                                    struct hif_driver_state_callbacks *cbk)
```

- Allocation of HIF instance
- Call appropriate bus open function

#### `hif_enable`

```
/***
 * hif_enable(): hif_enable
 * @hif_ctx: hif_ctx
 * @dev: dev
 * @bdev: bus dev
 * @bid: bus ID
 * @bus_type: bus type
```

```

* @type: enable type
*
* Return: QDF_STATUS
*/
QDF_STATUS hif_enable(struct hif_opaque_softc *hif_ctx, struct device
*dev,
void *bdev, const hif_bus_id *bid,
enum qdf_bus_type bus_type,
enum hif_enable_type type)

```

Call appropriate bus enable function.

Bus specific initialization

- Reading config space
- Configuring IRQ
- Configuring the h/w such as CE

### **hif\_disable**

void hif\_disable(struct hif\_opaque\_softc \*hif\_ctx, enum hif\_disable\_type type)

Operations involve calling of appropriate bus disable routine and disabling of interrupts.

#### **2.7.2.3 Fast path callback**

The earlier code of CE used to access the upper layer data structure htt\_pdev for providing information to upper layer. And used to call HTT functions directly.

The layering b/w CE and HTT has been cleaned by introducing appropriate callback and removing the access of other layer structure.

### **Callback registration**

```

/**
* hif_ce_fastpath_cb_register() - Register callback for fastpath msg handler
* @handler: Callback function
* @context: handle for callback function
*
* Return: QDF_STATUS_SUCCESS on success or QDF_STATUS_E_FAILURE
*/
int hif_ce_fastpath_cb_register(struct hif_opaque_softc *hif_ctx,
                               fastpath_msg_handler handler,
                               void *context)

```

## Callback function

```
typedef void (*fastpath_msg_handler)(void *, qdf_nbuf_t *, uint32_t);
```

### 2.7.3 HIF/CE Major data structures

The public data structure to HIF is an opaque data structure `hif_opaque_softc`. Internal data structure are classified in two categories

1. Bus independent data structure
2. Bus specific data structure

#### 2.7.3.1 Bus-independent data structure

This is a common data structure for all the buses. It does not contain any bus-specific information.

```
struct hif_softc {
    struct hif_opaque_softc osc;
    struct hif_config_info hif_config;
    struct hif_target_info target_info;
    void __iomem *mem;
    enum qdf_bus_type bus_type;
    struct hif_bus_ops bus_ops;
    void *ce_id_to_state[CE_COUNT_MAX];
    qdf_device_t qdf_dev;
    bool hif_init_done;
    bool request_irq_done;
    /* Packet statistics */
    struct hif_ce_stats pkt_stats;
    enum hif_target_status target_status;

    struct targetdef_s *targetdef;
    struct ce_reg_def *target_ce_def;
    struct hostdef_s *hostdef;
    struct host_shadow_regs_s *host_shadow_regs;

    bool recovery;
    bool notice_send;
    uint32_t ce_irq_summary;
    /* No of copy engines supported */
    unsigned int ce_count;
    atomic_t active_tasklet_cnt;
    atomic_t link_suspended;
    uint32_t *vaddr_rri_on_ddr;
    int linkstate_vote;
    bool fastpath_mode_on;
    atomic_t tasklet_from_intr;
    int htc_htt_tx_endpoint;
    qdf_dma_addr_t mem_pa;
    bool athdiag_procfs_initiated;

#ifdef FEATURE_NAPI
    struct qca_napi_data napi_data;
#endif /* FEATURE_NAPI */
    struct hif_driver_state_callbacks callbacks;
```

```

        uint32_t hif_con_param;
#define QCA_NSS_WIFI_OFFLOAD_SUPPORT
        uint32_t nss_wifi_ol_mode;
#endif
};

```

### 2.7.3.2 Bus-specific data structure

All the supported bus in HIF has their own independent data structure

The following is an example of the PCI bus:

```

struct hif_pci_softc {
    struct HIF_CE_state ce_sc;
    void __iomem *mem;           /* PCI address. */
    /* For efficiency, should be first in struct */

    struct device *dev;
    struct pci_dev *pdev;
    int num_msi_intrs;          /* number of MSI interrupts granted */
    /* 0 --> using legacy PCI line interrupts */
    struct tasklet_struct intr_tq; /* tasklet */

    struct hif_msi_info msi_info;
    int irq;
    int irq_event;
    int cacheline_sz;
    u16 devid;
    qdf_dma_addr_t soc_PCIE_bar0;
    struct hif_tasklet_entry tasklet_entries[HIF_MAX_TASKLET_NUM];
    bool pci_enabled;
    qdf_spinlock_t irq_lock;
    qdf_work_t reschedule_tasklet_work;
    uint32_t lcr_val;

#ifdef FEATURE_RUNTIME_PM
    atomic_t pm_state;
    uint32_t prevent_suspend_cnt;
    struct hif_pci_pm_stats pm_stats;
    struct work_struct pm_work;
    spinlock_t runtime_lock;
    struct timer_list runtime_timer;
    struct list_head prevent_suspend_list;
    unsigned long runtime_timer_expires;
    struct hif_pm_runtime_lock *prevent_linkdown_lock;
#endif
#ifdef WLAN_OPEN_SOURCE
    struct dentry *pm_dentry;
#endif
#endif
};

```

The following is an example of the USB bus:

```

struct hif_usb_softc {
    struct _HIF_DEVICE_USB hif_hdl;
    /* For efficiency, should be first in struct */
    struct device *dev;

```

```

        struct usb_dev *pdev;
        /*
         * Guard changes to Target HW state and to software
         * structures that track hardware state.
         */
        u16 devid;
        struct usb_interface *interface;
        struct notifier_block reboot_notifier; /* default mode before
reboot */
        u8 suspend_state;
        u8 *fw_data;
        u32 fw_data_len;
        /* structure to save FW RAM dump (Rome USB) */
        struct fw_ramdump *ramdump[FW_RAM_SEG_CNT];
        uint8_t rambump_index;
        bool fw_ram_dumping;
        /* enable FW self-recovery for Rome USB */
        bool enable_self_recovery;
    };

SDIO:
struct hif_sdio_softc {
    struct hif_softc ol_sc;
    struct device *dev;
    struct _NIC_DEV aps_osdev;
    struct tasklet_struct intr_tq; /* tasklet */

    int irq;
    /*
     * Guard changes to Target HW state and to software
     * structures that track hardware state.
     */
    spinlock_t target_lock;
    void *hif_handle;
    void *ramdump_base;
    unsigned long rambump_address;
    unsigned long rambump_size;
    struct targetdef_s *targetdef;
    struct hostdef_s *hostdef;
};


```

## 2.7.4 Concurrent support for USB/SIDO/PCI

HIF facilitates USB and PCI other WLAN chip to be connected simultaneously and the same HIF must be able to provide the service concurrently.

During `hif_bus_open`, the driver registers bus specific routine per device.

```

struct hif_bus_ops {
    QDF_STATUS (*hif_bus_open)(struct hif_softc *hif_sc,
                           enum qdf_bus_type bus_type);
    void (*hif_bus_close)(struct hif_softc *hif_sc);
    void (*hif_bus_prevent_linkdown)(struct hif_softc *hif_sc, bool
flag);

```

```

        void (*hif_reset_soc)(struct hif_softc *hif_sc);
        int (*hif_bus_suspend)(struct hif_softc *hif_ctx);
        int (*hif_bus_resume)(struct hif_softc *hif_ctx);
        int (*hif_target_sleep_state_adjust)(struct hif_softc *scn,
                                             bool sleep_ok, bool wait_for_it);
        void (*hif_disable_isr)(struct hif_softc *hif_sc);
        void (*hif_nointrs)(struct hif_softc *hif_sc);
        QDF_STATUS (*hif_enable_bus)(struct hif_softc *hif_sc,
                                     struct device *dev, void *bdev, const hif_bus_id
                                     *bid,
                                     enum hif_enable_type type);
        void (*hif_disable_bus)(struct hif_softc *hif_sc);
        int (*hif_bus_configure)(struct hif_softc *hif_sc);
        QDF_STATUS (*hif_get_config_item)(struct hif_softc *hif_sc,
                                         int opcode, void *config, uint32_t config_len);
        void (*hif_set_mailbox_swap)(struct hif_softc *hif_sc);
        void (*hif_claim_device)(struct hif_softc *hif_sc);
        void (*hif_shutdown_device)(struct hif_softc *hif_sc);
        void (*hif_stop)(struct hif_softc *hif_sc);
        void (*hif_cancel_deferred_target_sleep)(struct hif_softc *hif_
sc);
        void (*hif_irq_disable)(struct hif_softc *hif_sc, int ce_id);
        void (*hif_irq_enable)(struct hif_softc *hif_sc, int ce_id);
        int (*hif_dump_registers)(struct hif_softc *hif_sc);
        void (*hif_dump_target_memory)(struct hif_softc *hif_sc,
                                       void *ramdump_base,
                                       uint32_t address, uint32_t size);
        void (*hif_ipa_get_ce_resource)(struct hif_softc *hif_sc,
                                       qdf_dma_addr_t *sr_base_paddr,
                                       uint32_t *sr_ring_size,
                                       qdf_dma_addr_t *reg_paddr);
        void (*hif_mask_interrupt_call)(struct hif_softc *hif_sc);
        void (*hif_enable_power_management)(struct hif_softc *hif_ctx,
                                           bool is_packet_log_enabled);
        void (*hif_disable_power_management)(struct hif_softc *hif_ctx);
        void (*hif_display_stats)(struct hif_softc *hif_ctx);
        void (*hif_clear_stats)(struct hif_softc *hif_ctx);
    };
}

```

The following are the specific routines that are called by the generic API though bus operations:

```

int hif_bus_configure(struct hif_softc *hif_sc)
{
    return hif_sc->bus_ops.hif_bus_configure(hif_sc);
}

```

## 2.7.5 Register table support for different chips

Register macros are stored in a structure and populated dynamically per target architecture at compile time. This structure is used to access the register offsets.

Converged register structures are in reg\_struct.h file. The following are the major structures:

- "struct targetdef\_s
  - "struct hostdef\_s
  - "struct host\_shadow\_regs\_s
  - "struct ce\_reg\_def

For each target, a <TARGET>def.c file is present, which initializes its target register definition structure by including corresponding header files. This target specific file has to define MY\_TARGET\_DEF, MY\_HOST\_DEF, MY\_CEREG\_DEF for initializing targetdef, hostdef and cereg\_def respectively. If any of these defines are not defined, these structures are not initialized.

Targetdef.c file will include all the required header files and will define required defines for initializing its register structures. After that it will include targetdef.h and hostdef.h header files. These header files, in turn, include target\_reg\_init.h and host\_reg\_init.h header files respectively. These header files will allocate global structures and initializes them with the register definitions of that target.

The following is an example of the register table initialization:

ar900Bdef.c is the target specific register table initialization file. This will include all the hardware header files as follows:

```
#include "AR900B/extrahw/soc_core_reg.h"  
#include "AR900B/hw/soc_pcie_reg.h"  
#include "AR900B/extrahw/ce_req_csr.h" ..
```

After including the header files, it initializes required the define functions as follow

```
define MY_TARGET_DEF AR900B_TARGETdef
#define MY_HOST_DEF AR900B_HOSTdef
                #include "targetdef.h"
targetdef.h  inturn includes target req init.h.
```

The target\_reg\_init.h header file allocates the structure by using the MY\_TARGET\_DEF macro as follows:

```
struct targetdef s *MY TARGET DEF = &my target def;
```

The target reg init.h header file initializes members as follows:

```
static struct targetdef_s my_target_def = {
    .d_RTC_SOC_BASE_ADDRESS = RTC_SOC_BASE_ADDRESS,
    .d_RTC_WMAC_BASE_ADDRESS = RTC_WMAC_BASE_ADDRESS, ...
```

## 2.7.6 NSS WLAN offload support

NSS WLAN offload function used to access most of the HIF data structure to provide information to the NSS. The direct access to the HIF data structure is removed and a cleaner API is provided to extract HIF information

The following new APIs are introduced:

- ```
1. struct hif_pipe_addl_info *hif_get_addl_pipe_info(struct hif_opaque_softc *osc,  
        struct hif_pipe_addl_info *hif_info, uint32_t pipe)
```

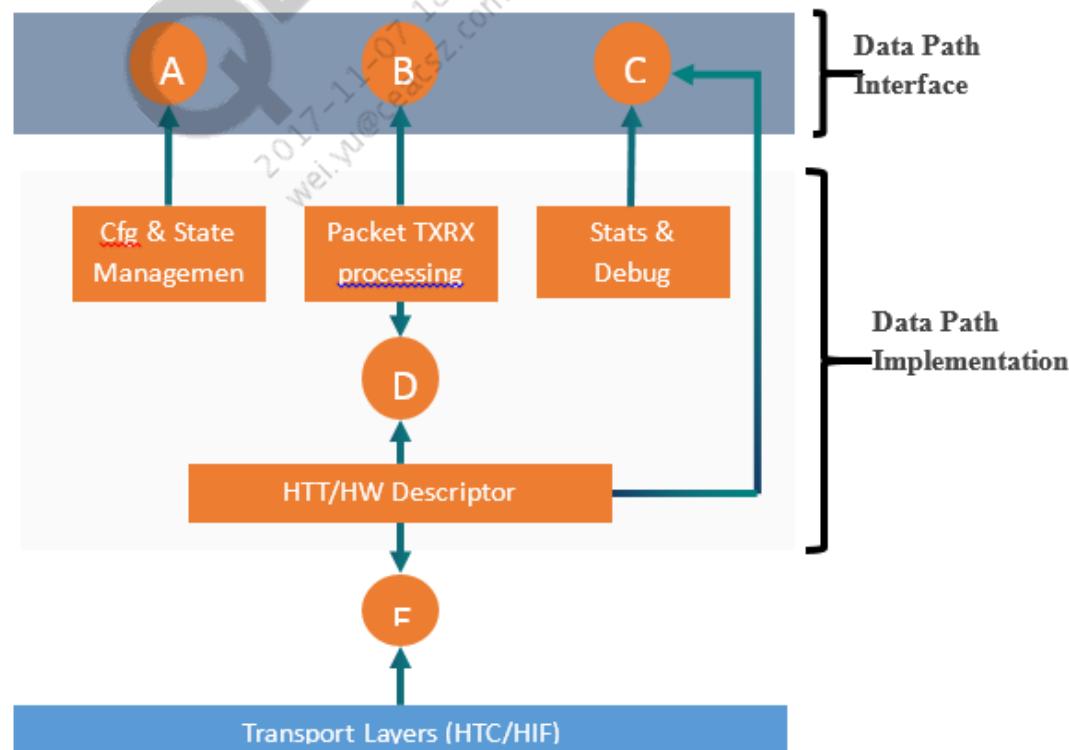
2. `uint32_t hif_set_nss_wifiol_mode(struct hif_opaque_softc *osc,  
                  uint32_t pipe_num);`
3. `int32_t hif_get_nss_wifiol_bypass_nw_process(struct hif_opaque_softc *osc);`

## 2.8 Plugin of Data Path Interface

The data path module provides transmit and receive functionalities. The implementation can depends on radio architecture and it differs between two families of radio chipsets. The data path module need to be modularized in a way that only required APIs made to public. The radio specific differences in implementation would be handled within the CDP module. Other modules use public interface APIs to invoke the transmit/receive functionality. The public interface APIs should be same for all radio families.

This section describes the design of Data path module and details of interface APIs to be used by all radio families of QCA chipsets. The plugin of Data path interface layer provides framework for easy plug-in of data path for future chipsets.

### 2.8.1 Host architecture



The data path modularization aims at the following:

- Identifying the data path APIs that invoked by modules outside of the data path on all family of drivers
- Separating the APIs identified into two groups:
  - a. Common APIs - these APIs are common for all family of radio drivers.
  - b. Driver specific APIs - these APIs belong to features that are specific to radio. The APIs are subdivided into three categories:
    - i. State management and configuration
    - ii. Transmit and Receive
    - iii. Statistics and Debugging

This effort also involves refactor legacy implementation to adhere to common data path interfaces. This change includes legacy implementation of chipsets such as QCA9984, QCA9980, QCA6174, and QCA6164. Two different instances of the data path implementation for legacy chipsets are resulted. OSIF, OL\_IF are refactored to invoke Common Data Path Interface. Legacy chipsets are refactored in data path implementation to adhere to Common Data Path Interface

The Host-Firmware interaction is not modified.

## 2.8.2 Dependency on other modules

Until QCA\_Networking\_2016\_SPF.3.0, ADF APIs are used because with the move to QDF, the changes have a dependency on QDF module. In addition to this, all external modules that call into the data path public APIs needed to be changed to adhere to the new CDP APIs.

## 2.8.3 Host low-level design

Based on the analysis of core data path module implementation of all drivers, the APIs can be categorized into three submodules:

1. Common APIs
2. Control APIs
3. Radio specific APIs

Though all drivers use same APIs, the implementation is different.

As part of this, the plan is to make core data path module at interface level only, all drivers use their own implementations.

Making APIs common at the interface level would help to have a single implementation of core data path for upcoming chipsets, and single driver can be used for all chipsets.

As part of this, the core data path is modularized such a way that the other modules in Wi-Fi drivers can use only public APIs to invoke data path operation. The internal APIs cannot be accessed by other modules.

### 2.8.3.1 Common APIs

The core data path common APIs are categorized into three groups. They are,

1. Control APIs
2. Tx/Rx APIs
3. Debug and statistics APIs

### 2.8.3.2 Control APIs

The control APIs provide functionality to allocate, free, initialization and registration of core data path data structures.

**Allocate pdev object for CDP module, this is allocated on radio initialization. Before perform any operations on CDP modules, pdev must be allocated and initialized**

```
ol_txrx_pdev_handle ol_txrx_pdev_attach(ol_pdev_handle ctrl_pdev, HTC_HANDLE htc_pdev,
qdf_device_t osdev);
```

**Parameters:**

- scn—radio handle, parent structure for radio
- HTC\_HANDLE—HTC module handle
- qdf\_device\_t—OS dev object

Returns the new pdev object for success, NULL for failure

**Frees the pdev object, it is done on removing radio/module unload**

```
void ol_txrx_pdev_detach(ol_txrx_pdev_handle pdev, int force);
```

**Parameters:**

- pdev—pdev object
- force—pass 1 to free all peers forcefully

Returns nothing

**Attaches HTT with target by sending configuration of the H2T ring to target.**

```
int ol_txrx_pdev_attach_target(ol_txrx_pdev_handle pdev);
```

**Parameters:**

ol\_txrx\_pdev\_handle—pdev pointer

Returns A\_OK for success; otherwise, returns failure

**Allocates vdev object in CDP modules and initializes vdev, and adds vdev object into vdev list**

```
ol_txrx_vdev_handle ol_txrx_vdev_attach(ol_txrx_pdev_handle pdev, uint8_t *vdev_mac_addr,
   uint8_t vdev_id, enum wlan_op_mode op_mode);
```

**Parameters:**

- ol\_txrx\_pdev\_handle—pdev pointer
- vdev\_mac\_addr—MAC address of the interface/vap
- vdev\_id—unique vdev id
- wlan\_op\_mode—operating mode

Returns the new vdev object

**Resets and frees vdev object in CDP module**

```
void ol_txrx_vdev_detach(ol_txrx_vdev_handle vdev,
                         ol_txrx_vdev_delete_cb callback, void *cb_context);
```

**Parameters:**

- ol\_txrx\_vdev\_handle—vdev object to be deleted
- ol\_txrx\_vdev\_delete\_cb—callback handler to invoke after vdev deletion
- cb\_context—arguments to pass on callback invoking

Does not return anything

**Allocates and initializes peer object then adds to vdev peer list. To enable traffic on station, the peer should be allocated.**

```
ol_txrx_peer_handle ol_txrx_peer_attach(ol_txrx_vdev_handle vdev, uint8_t *peer_mac_addr);
```

**Parameters:**

- vdev—vdev object to which this peer is associated.
- peer\_mac\_addr—MAC address of peer

Returns new peer object on success, otherwise returns NULL

**Frees the peer object, this needs to be done station disconnection**

```
void ol_txrx_peer_detach(ol_txrx_peer_handle peer);
```

**Parameters:**

- peer—peer handle

Returns nothing

### **Configures the vdev as monitor mode vdev**

```
int ol_txrx_set_monitor_mode(ol_txrx_vdev_handle vdev);
```

#### **Parameters:**

- vdev—vdev\_handle

Returns 0 for success, -1 for failure

### **Configure monitor mode channel**

```
void ol_txrx_set_curchan(ol_txrx_pdev_handle pdev, uint32_t chan_mhz);
```

#### **Parameters:**

- pdev—pdev\_handle
- chan\_mhz—freq

Returns nothing

### **Sets the privacy filters for vdev to allow specific protocol frames before key derivation**

```
void ol_txrx_set_privacy_filters(ol_txrx_vdev_handle vdev, void *filter, uint32_t num);
```

#### **Parameters:**

- vdev—vdev\_handle
- filter—filter data
- num—num of filters

Returns nothing

### **Register Tx/Rx APIs for vdev to invoke them for Tx/Rx traffic.**

```
void ol_txrx_vdev_register(ol_txrx_vdev_handle vdev, void *osif_vdev, struct ol_txrx_ops *txrx_ops);
```

#### **Parameters:**

- vdev—vdev object
- txrx\_ops—pointer to structure which passes the handlers to CDP module

Returns nothing

### **Helper APIs to get the data from vdev**

```
uint8_t *ol_txrx_get_vdev_mac_addr(ol_txrx_vdev_handle vdev);
```

```
struct qdf_mac_addr *ol_txrx_get_vdev_struct_mac_addr(ol_txrx_vdev_handle vdev);
```

```
ol_txrx_pdev_handle ol_txrx_get_pdev_from_vdev(ol_txrx_vdev_handle vdev);
```

```
ol_pdev_handle ol_txrx_get_ctrl_pdev_from_vdev(ol_txrx_vdev_handle vdev);
```

**Parameters:**

- *ol\_txrx\_vdev\_handle vdev*

Returns appropriate data

### 2.8.3.3 Tx/Rx APIs

These Data Tx/Rx APIs implement data/mgmt. Tx/Rx APIs.

#### Transmits the management frame to FW

```
int ol_txrx_mgmt_send(ol_txrx_vdev_handle vdev, qdf_nbuf_t tx_mgmt_frm, uint8_t type);
```

**Parameters:**

- *ol\_txrx\_vdev\_handle*—vdev object
- *qdf\_nbuf\_t*—frame buffer
- *type*—frame type (Ethernet or RAW or Mgmt)

Returns 0 on success, negative value on failure.

#### Transmit management frame to FW with tx rate (of 6mbps) and user configured frequency

```
int ol_txrx_mgmt_send_ext(ol_txrx_vdev_handle vdev, qdf_nbuf_t tx_mgmt_frm, uint8_t type, uint8_t use_6mbps, uint16_t chanfreq);
```

**Parameters:**

- *ol\_txrx\_vdev\_handle*—vdev object
- *qdf\_nbuf\_t*—frame buffer
- *type*—frame type (Ethernet or RAW or Mgmt)
- *use\_6mbps*—to send frame at 6 mbps
- *chanfreq*—channel frequency

Returns 0 on success, negative value on failure.

#### Callback handlers for mgmt/data Tx frames

*ol\_txrx\_mgmt\_tx\_cb - tx management delivery notification callback function*

*typedef void*

```
(*ol_txrx_mgmt_tx_cb)(void *ctxt, qdf_nbuf_t tx_mgmt_frm, int had_error);
```

```
void ol_txrx_mgmt_tx_cb_set(ol_txrx_pdev_handle pdev, uint8_t type, ol_txrx_mgmt_tx_cb download_cb, ol_txrx_mgmt_tx_cb ota_ack_cb, void *ctxt);
```

```

int ol_txrx_get_tx_pending(ol_txrx_pdev_handle pdev);

ol_txrx_data_tx_cb - Function registered with the data path
typedef void
(*ol_txrx_data_tx_cb)(void *ctxt, qdf_nbuf_t tx_frm, int had_error);

void
ol_txrx_data_tx_cb_set(ol_txrx_vdev_handle data_vdev,
                       ol_txrx_data_tx_cb callback, void *ctxt);

```

#### 2.8.3.4 Debug and Statistics API

These APIs to configure and request Tx/Rx stats from FW and prints them on console.

```

int ol_txrx_aggr_cfg(ol_txrx_vdev_handle vdev, int max_subfrms_ampdu, int max_subfrms_
amsdu);

int ol_txrx_fw_stats_get(ol_txrx_vdev_handle vdev, struct ol_txrx_stats_req *req,
                        bool response_expected);

int ol_txrx_debug(ol_txrx_vdev_handle vdev, int debug_specs);

void ol_txrx_fw_stats_cfg(ol_txrx_vdev_handle vdev, uint8_t cfg_stats_type,
                          uint32_t cfg_val);

void ol_txrx_print_level_set(unsigned level);

```

#### 2.8.4 Driver-specific APIs

The driver-specific APIs are divided into the following categories:

1. cdp\_txrx\_ctrl.h - Vdev/pdev/peer parameter configuration and helper APIs
2. cdp\_txrx\_host\_stats.h - APIs for Tx/Rx debug statistics
3. cdp\_txrx\_stats\_struct.h - Data structures and parameter definition for statistics
4. cdp\_txrx\_me.h - Multicast enhancement-related APIs
5. cdp\_txrx\_mon.h - Monitor mode-related APIs
6. cdp\_txrx\_pflow.h - Peer flow control
7. cdp\_txrx\_raw.h - Raw mode APIs

8. cdp\_txrx\_wds.h - WDS APIs
9. cdp\_txrx\_bus.h - Bus related operations
10. cdp\_txrx\_flow\_ctrl\_legacy.h - Legacy flow control with pause/unpause mechanism and per vdev queues
11. cdp\_txrx\_flow\_ctrl\_v2.h - New flow control (version 2)
12. cdp\_txrx\_ipa.h - IP accelerator related
13. cdp\_txrx\_lro.h - Large receive offload related
14. cdp\_txrx\_mon.h - Monitor mode APIs
15. cdp\_txrx\_peer\_ops.h - peer specific operations
16. cdp\_txrx\_pmf.h - Protected Management Frame
17. cdp\_txrx\_tx\_throttle.h - Transmit path throttling APIs

Refer to the header files for API details.

### 2.8.5 Changes to non-public interfaces

The following table describes the driver specific changes:

| Old API                                                      | New API                                                                                                                                 |
|--------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| void ol_tx_flush_buffers_pfctrl(struct ol_txrx_pdev_t *pdev) | void ol_tx_flush_buffers_pfctrl(struct ol_txrx_vdev_t *vdev)<br>pdev can be accessed from vdev, vdev can be retrieved from osdev handle |

### 2.8.6 Changes to public interfaces

The following APIs are changed as non-public interfaces because they are not used outside:

| Old API                                                                                                                                                              | New API                     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|
| void ol_rx_err(ol_pdev_handle pdev, u_int8_t vdev_id, u_int8_t *peer_mac_addr, int tid, u_int32_t tsf32, enum ol_rx_err_type err_type, qdf_nbuf_t rx_frame)          | Moved from CDP header files |
| int ol_rx_notify(ol_pdev_handle pdev, u_int8_t vdev_id, u_int8_t *peer_mac_addr, int tid, u_int32_t tsf32, enum ol_rx_notify_type notify_type, qdf_nbuf_t rx_frame); |                             |
| inline int ol_tx_tso_process_skb (ol_txrx_vdev_handle vdev,qdf_nbuf_t msdu)                                                                                          | Moved from CDP header files |
| inline int ol_tx_sg_process_skb (ol_txrx_vdev_handle vdev,qdf_nbuf_t msdu)                                                                                           | Moved inside CDP            |

| Old API                                                                                                                                                         | New API                                                                                                                                                       |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>int ol_tx_ll_pflow_ctrl (ol_txrx_vdev_handle vdev, qdf_nbuf_t netbuf)</code>                                                                              | Moved inside CDP                                                                                                                                              |
| PEER FLOW CONTROL:<br><code>int ol_tx_ll_cachedhdr(ol_txrx_vdev_handle vdev, qdf_nbuf_t netbuf,uint16_t peer_id, uint8_t tid)</code>                            | Moved inside CDP                                                                                                                                              |
| Non Peer flow control:<br><code>int ol_tx_ll_cachedhdr(ol_txrx_vdev_handle vdev, qdf_nbuf_t netbuf)</code>                                                      |                                                                                                                                                               |
| PEER FLOW CONTROL:<br><br><code>uint32_t ol_tx_ll_fast(ol_txrx_vdev_handle vdev, qdf_nbuf_t *nbuf_arr, uint32_t num_msdu, uint16_t peer_id, uint8_t tid)</code> | Moved inside CDP                                                                                                                                              |
| Non Peer flow control:<br><code>uint32_t ol_tx_ll_fast(ol_txrx_vdev_handle vdev, qdf_nbuf_t *nbuf_arr, uint32_t num_msdu)</code>                                |                                                                                                                                                               |
| <code>ol_txrx_peer_handle ol_txrx_peer_attach( ol_txrx_pdev_handle pdev, ol_txrx_vdev_handle vdev, u_int8_t *peer_mac_addr)</code>                              | <code>ol_txrx_peer_handle ol_txrx_peer_attach(ol_txrx_vdev_handle vdev, u_int8_t *peer_mac_addr)</code>                                                       |
| <code>int ol_txrx_set_monitor_mode_vap( ol_txrx_pdev_handle pdev, ol_txrx_vdev_handle vdev);</code>                                                             | <code>int ol_txrx_set_monitor_mode(ol_txrx_vdev_handle vdev);</code>                                                                                          |
| <code>void ol_txrx_osif_vdev_register(ol_txrx_vdev_handle vdev, void *osif_vdev, struct ol_txrx_ops *txrx_ops);</code>                                          | <code>void ol_txrx_vdev_register(ol_txrx_vdev_handle vdev, void *osif_vdev, struct ol_txrx_ops *txrx_ops);</code>                                             |
| <code>int ol_txrx_fw_stats_get(ol_txrx_vdev_handle vdev, struct ol_txrx_stats_req *req);</code>                                                                 | <code>int ol_txrx_fw_stats_get(ol_txrx_vdev_handle vdev, struct ol_txrx_stats_req *req, bool response_expected);</code>                                       |
|                                                                                                                                                                 | <code>int ol_txrx_reset_monitor_mode (ol_txrx_pdev_handle pdev);</code>                                                                                       |
|                                                                                                                                                                 | <code>void ol_tx_flush_buffers(struct ol_txrx_vdev_t *vdev);</code>                                                                                           |
| <code>void ol_txrx_mgmt_tx_cb_set(ol_txrx_pdev_handle pdev, u_int8_t type, ol_txrx_mgmt_tx_cb cb, void *ctxt);</code>                                           | <code>void ol_txrx_mgmt_tx_cb_set( ol_txrx_pdev_handle pdev, u_int8_t type, ol_txrx_mgmt_tx_cb download_cb, ol_txrx_mgmt_tx_cb ota_ack_cb, void *ctxt)</code> |

| Old API                                                                                                                                                                                                                                                                                                                                                                    | New API                                                                                                                                                                                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>struct ol_txrx_vdev_t {     ...     ol_rx_check_wai_fp osif_check_wai; }</pre>                                                                                                                                                                                                                                                                                        | <pre>struct ol_txrx_vdev_t {     ...     ol_txrx_rx_check_wai_fp osif_check_wai; }</pre>                                                                                                                                                                                                                                                 |
| <pre>struct ol_txrx_osif_ops {     /* tx function pointers - specified by txrx, stored by      * OS shim */     struct {         ol_txrx_tx_fp      std;         ol_txrx_tx_non_std_fp non_std;     } tx;     /* rx function pointers - specified by OS shim, stored      * by txrx */     struct {         ol_txrx_rx_fp      std;         ...     } rx;     ... };</pre> | <pre>struct ol_txrx_ops {     /* tx function pointers - specified by txrx, stored by      * OS shim */     struct {         ol_txrx_tx_fp      tx;         ...     } tx;     /* rx function pointers - specified by OS shim,      * stored by txrx */     struct {         ol_txrx_rx_fp      rx;         ...     } rx;     ... };</pre> |

1. To use data path stats structures from other modules, `cdp_txrx_stats_struct.h` must be included. Previously, until QCA\_Networking\_2016.SPF.3.0, `ol_params.h` was used as header file, which is currently removed.
2. OS layer used to invoke separate CDP module calls for TSO, SG, data frame. With the changes, OS layer invokes a single Tx callback for all types of Tx frames. The CDP module classifies the frame and invokes appropriate APIs.
3. Tx Flush buffers code was outside of CDP module, it is moved inside CDP.
4. As part of this move, the unnecessary inclusion of CDP header files is removed.
5. With the change in the registration mechanism for the per vdev transmit and receive functions, the following changes were needed:

Receive Function:

Register a per-vdev receive callback function using the vdev registration API provided by the converged data path API. The changes include

- removing the older method of registering the receive callback
- modifying the HDD callback to adhere to the rx callback function defined by the converged data path API
- modifying the invocation of the callback function in the data path

Transmit Function:

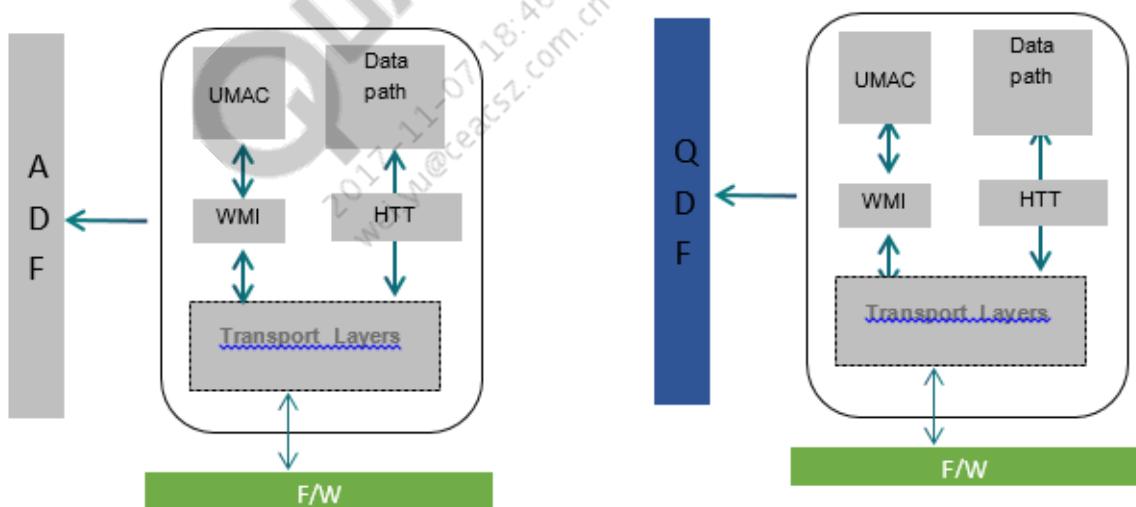
Register a per-vdev transmit function during vdev registration. The OS interface (HDD) adapter stores this transmit function and invokes it to transmit data frames.

6. IPA—Changed the IPA invocations to pass a structure containing the configuration parameters.
7. Peer Operations—There were several violations identified where modules outside of the data path were accessing and modifying data path specific peer objects. Identified these and introduces APIs for peer-related invocations.

## 2.9 Optimization of OS abstraction layer

There is a need for cleaner logical separation of OS Abstraction layer from the driver. ADF comprises of many unused APIs, macros, and bloated SKB CB. All these parameters are effectively organized and re-structured resulting in QDF layer.

### 2.9.1 ADF to QDF transition architecture



The following enhancements are made for the transition from ADF to QDF:

- QDF abstraction layer is cleaned up to remove unused APIs. Examples are Linux specific procfs APIs, and unused bus and IO APIs.
- Naming convention for QDF APIs and Macros is modified to have qdf\_tag (instead of adf\_)
- Directory structure of qdf is improved with cleaner hierarchy of "inc" and "Linux" subfolders to hold relevant files.

- Some of the data structures are reorganized (cleanup of unused members). Examples are skb cb and device handle structures.
- Device, chip, or OS-specific code that cannot be abstracted is moved out of QDF into OS specific folders.

## 2.9.2 List of ADF and QDF APIs

The following table provides a mapping of the ADF APIs and corresponding QDF APIs:

| ADF API                        | QDF API                            |
|--------------------------------|------------------------------------|
| adf_os_str_len                 | qdf_str_len                        |
| adf_os_mem_alloc               | qdf_mem_malloc                     |
| adf_os_mem_alloc_outline       | qdf_mem_alloc_outline              |
| adf_os_mem_free                | qdf_mem_free                       |
| adf_os_mem_free_outline        | qdf_mem_free_outline               |
| adf_os_mem_alloc_consistent    | qdf_mem_alloc_consistent           |
| adf_os_mem_free_consistent     | qdf_mem_free_consistent            |
| adf_os_mem_copy                | qdf_mem_copy                       |
| adf_os_mem_move                | qdf_mem_move                       |
| adf_os_mem_set                 | qdf_mem_set                        |
| adf_os_mem_zero                | qdf_mem_zero                       |
| adf_os_mem_zero_outline        | qdf_mem_zero_outline               |
| adf_os_mem_cmp                 | qdf_mem_cmp                        |
| adf_os_str_cmp                 | qdf_str_cmp                        |
| adf_os_str_ncopy               | qdf_str_ncopy                      |
| adf_os_mem_map_nbytes_single   | qdf_mem_map_nbytes_single          |
| adf_os_mem_unmap_nbytes_single | qdf_mem_unmap_nbytes_single        |
| adf_mempool_init               | qdf_mempool_init                   |
| adf_mempool_destroy            | qdf_mempool_destroy                |
| adf_mempool_alloc              | qdf_mempool_alloc                  |
| adf_mempool_free               | qdf_mempool_free                   |
| adf_mempool_t                  | qdf_mempool_t                      |
|                                | qdf_mem_clean                      |
|                                | qdf_mem_init                       |
|                                | qdf_mem_exit                       |
|                                | qdf_mem_malloc_debug               |
|                                | qdf_mem_dma_sync_single_for_device |
|                                |                                    |
| adf_os_init_mutex              | qdf_mutex_init                     |

| ADF API                        | QDF API                       |
|--------------------------------|-------------------------------|
| adf_os_mutex_acquire           | qdf_mutex_acquire             |
| adf_os_mutex_acquire_intr      | qdf_semaphore_acquire_intr    |
| adf_os_mutex_release           | qdf_mutex_release             |
| adf_os_sem_t                   | qdf_semaphore_t               |
| adf_os_spinlock_init           | qdf_spinlock_init             |
| adf_os_spinlock_destroy        | qdf_spinlock_destroy          |
| adf_os_spin_trylock_bh         | qdf_spin_trylock_bh           |
| adf_os_spin_trylock_bh_outline | qdf_spin_trylock_bh_outline   |
| adf_os_spin_lock_bh            | qdf_spin_lock_bh              |
| adf_os_spin_lock_bh_outline    | qdf_spin_lock_bh_outline      |
| adf_os_spin_unlock_bh          | qdf_spin_unlock_bh            |
| adf_os_spin_unlock_bh_outline  | qdf_spin_unlock_bh_outline    |
| adf_os_spinlock_t              | qdf_spinlock_t                |
| adf_os_mutex_t                 | qdf_mutex_t                   |
| adf_os_spin_lock               | qdf_spin_lock                 |
| adf_os_spin_unlock             | qdf_spin_unlock               |
| adf_os_spin_lock_irq           | qdf_spin_lock_irq             |
| adf_os_spin_unlock_irq         | qdf_spin_unlock_irq           |
| adf_os_mutex_acquire_timeout   | qdf_semaphore_acquire_timeout |
|                                | qdf_semaphore_init            |
|                                | qdf_semaphore_acquire         |
|                                | qdf_semaphore_release         |
|                                | qdf_wake_lock_init            |
|                                | qdf_wake_lock_acquire         |
|                                | qdf_wake_lock_timeout_acquire |
|                                | qdf_wake_lock_release         |
|                                | qdf_wake_lock_destroy         |
|                                | qdf_spinlock_acquire          |
|                                | qdf_spinlock_release          |
|                                | qdf_spin_lock_irqsave         |
|                                | qdf_spin_unlock_irqrestore    |
|                                | qdf_mutex_destroy             |
|                                |                               |
| adf_os_time_t                  | qdf_time_t                    |
| adf_os_ticks                   | qdf_system_ticks              |
| adf_os_ticks_to_msecs          | qdf_system_ticks_to_msecs     |
| adf_os_msecs_to_ticks          | qdf_system_msecs_to_ticks     |

| ADF API                  | QDF API                        |
|--------------------------|--------------------------------|
| adf_os_getuptime         | qdf_get_system_uptime          |
| adf_os_gettimestamp      | qdf_get_system_timestamp       |
| adf_os_udelay            | qdf_udelay                     |
| adf_os_mdelay            | qdf_mdelay                     |
| adf_os_time_after        | qdf_system_time_after          |
| adf_os_time_before       | qdf_system_time_before         |
| adf_os_time_after_eq     | qdf_system_time_after_eq       |
|                          |                                |
| adf_os_timer_t           | qdf_softirq_timer_t            |
| adf_os_timer_init        | qdf_softirq_timer_init         |
| adf_os_timer_start       | qdf_softirq_timer_start        |
| adf_os_timer_mod         | qdf_softirq_timer_mod          |
| adf_os_timer_cancel      | qdf_softirq_timer_cancel       |
| adf_os_timer_sync_cancel | qdf_timer_sync_cancel          |
| adf_os_timer_free        | qdf_softirq_timer_free         |
|                          | qdf_timer_module_init          |
|                          | qdf_mc_timer_manager_init      |
|                          | qdf_timer_clean                |
|                          | qdf_mc_timer_exit              |
|                          | qdf_mc_timer_init_debug        |
|                          | qdf_mc_timer_init              |
|                          | qdf_mc_timer_destroy           |
|                          | qdf_mc_timer_start             |
|                          | qdf_mc_timer_stop              |
|                          | qdf_mc_timer_get_system_ticks  |
|                          | qdf_mc_timer_get_system_time   |
|                          | qdf_get_monotonic_boottime     |
|                          | qdf_get_log_timestamp          |
|                          | qdf_linux_timer_callback       |
|                          | qdf_mc_timer_get_current_state |
| adf_net_alloc_wlanunit   | qdf_net_alloc_wlanunit         |
| adf_net_delete_wlanunit  | qdf_net_delete_wlanunit        |
| adf_net_ifc_name2unit    | qdf_net_ifc_name2unit          |
| adf_net_new_wlanunit     | qdf_net_new_wlanunit           |
| adf_os_ntohl             | qdf_ntohl                      |
| adf_os_htons             | qdf_htons                      |
| adf_os_htonl             | qdf_htonl                      |

| ADF API            | QDF API                          |
|--------------------|----------------------------------|
| adf_os_cpu_to_le64 | qdf_cpu_to_le64                  |
|                    |                                  |
|                    | qdf_list_insert_front            |
|                    | qdf_list_insert_back_size        |
|                    | qdf_list_remove_front            |
|                    | qdf_list_peek_next               |
|                    | qdf_list_init                    |
|                    | qdf_list_destroy                 |
|                    | qdf_list_size                    |
|                    | qdf_list_insert_back             |
|                    | qdf_list_remove_back             |
|                    | qdf_list_peek_front              |
|                    | qdf_list_remove_node             |
|                    | qdf_list_empty                   |
|                    |                                  |
|                    | qdf_event_init                   |
|                    | qdf_event_set                    |
|                    | qdf_event_reset                  |
|                    | qdf_event_destroy                |
|                    | qdf_wait_single_event            |
|                    |                                  |
|                    | qdf_trace_set_level              |
|                    | qdf_trace_get_level              |
|                    | qdf_trace_set_module_trace_level |
|                    | qdf_snprintf                     |
|                    | qdf_trace_msg                    |
|                    | qdf_trace_hex_dump               |
|                    | qdf_trace_get_level              |
|                    | qdf_trace                        |
|                    | qdf_trace_register               |
|                    | qdf_trace_spin_lock_init         |
|                    | qdf_trace_init                   |
|                    | qdf_trace_enable                 |
|                    | qdf_trace_dump_all               |
|                    | qdf_dp_trace_spin_lock_init      |
|                    | qdf_dp_trace_init                |
|                    | qdf_dp_trace_set_value           |

| ADF API                     | QDF API                   |
|-----------------------------|---------------------------|
|                             | qdf_dp_trace_set_track    |
|                             | qdf_dp_trace              |
|                             | qdf_dp_trace_dump_all     |
|                             | qdf_dp_display_record     |
|                             | qdf_dp_trace_enable_track |
|                             | qdf_trace_set_value       |
|                             | qdf_trace_display         |
|                             |                           |
| adf_os_perf_init            | qdf_perf_init             |
| adf_os_perf_destroy         | qdf_perf_destroy          |
| adf_os_perf_start           | qdf_perf_start            |
| adf_os_perf_end             | qdf_perf_end              |
| adf_os_perfmod_init         | qdf_perfmod_init          |
| adf_os_perfmod_exit         | qdf_perfmod_exit          |
|                             |                           |
|                             |                           |
| adf_os_unlikely             | qdf_unlikely              |
| adf_os_likely               | qdf_likely                |
| adf_os_mb                   | qdf_mb                    |
| adf_os_min                  | qdf_min                   |
| adf_os_max                  | qdf_max                   |
| adf_os_assert               | qdf_assert                |
| adf_os_assert_always        | qdf_assert_always         |
| adf_os_target_assert_always | qdf_target_assert_always  |
|                             | qdf_status_to_os_return   |
|                             | qdf_in_interrupt          |
|                             | qdf_container_of          |
|                             | qdf_is_macaddr_equal      |
|                             | qdf_is_macaddr_zero       |
|                             | qdf_zero_macaddr          |
|                             | qdf_is_macaddr_group      |
|                             | qdf_is_macaddr_broadcast  |
|                             | qdf_copy_macaddr          |
|                             | qdf_set_macaddr_broadcast |
|                             | qdf_set_u16               |
|                             | qdf_get_u16               |
|                             | qdf_get_u32               |

| ADF API                    | QDF API                 |
|----------------------------|-------------------------|
| adf_os_ntohs               | qdf_ntohs               |
| adf_os_ntohl               | qdf_ntohl               |
| adf_os_htons               | qdf_htons               |
| adf_os_htonl               | qdf_htonl               |
| adf_os_cpu_to_le16         | qdf_cpu_to_le16         |
| adf_os_cpu_to_le32         | qdf_cpu_to_le32         |
| adf_os_cpu_to_le64         | qdf_cpu_to_le64         |
| adf_os_be16_to_cpu         | qdf_be16_to_cpu         |
| adf_os_be32_to_cpu         | qdf_be32_to_cpu         |
| adf_os_be64_to_cpu         | qdf_be64_to_cpu         |
| adf_os_le16_to_cpu         | qdf_le16_to_cpu         |
| adf_os_le32_to_cpu         | qdf_le32_to_cpu         |
| adf_os_le64_to_cpu         | qdf_le64_to_cpu         |
|                            |                         |
| adf_os_atomic_t            | qdf_atomic_t            |
| adf_os_atomic_init         | qdf_atomic_init         |
| adf_os_atomic_read         | qdf_atomic_read         |
| adf_os_atomic_inc          | qdf_atomic_inc          |
| adf_os_atomic_dec          | qdf_atomic_dec          |
| adf_os_atomic_add          | qdf_atomic_add          |
| adf_os_atomic_dec_and_test | qdf_atomicdec_and_test  |
| adf_os_atomic_set          | qdf_atomic_set          |
|                            | qdf_atomic_inc_return   |
|                            |                         |
|                            |                         |
| adf_os_work_t              | qdf_work_t              |
| adf_os_delayed_work_t      | qdf_delayed_work_t      |
| adf_os_workqueue_t         | qdf_workqueue_t         |
| adf_os_bh_t                | qdf_bh_t                |
| adf_os_create_bh           | qdf_create_bh           |
| adf_os_sched_bh            | qdf_sched_bh            |
| adf_os_destroy_bh          | qdf_destroy_bh          |
| adf_os_create_work         | qdf_create_work         |
| adf_os_create_delayed_work | qdf_create_delayed_work |
| adf_os_create_workqueue    | qdf_create_workqueue    |
| adf_os_queue_work          | qdf_queue_work          |
| adf_os_queue_delayed_work  | qdf_queue_delayed_work  |

| ADF API                  | QDF API                  |
|--------------------------|--------------------------|
| adf_os_flush_workqueue   | qdf_flush_workqueue      |
| adf_os_destroy_workqueue | qdf_destroy_workqueue    |
| adf_os_sched_work        | qdf_sched_work           |
| adf_os_flush_work        | qdf_flush_work           |
| adf_os_disable_work      | qdf_disable_work         |
| adf_os_destroy_work      | qdf_destroy_work         |
|                          |                          |
|                          |                          |
| adf_net_udphdr_t         | qdf_net_udphdr_t         |
| adf_net_dhcp(hdr_t       | qdf_net_dhcp(hdr_t       |
| adf_net_vid_t            | qdf_net_vid_t            |
| adf_net_dev_info_t       | qdf_net_dev_info_t       |
| adf_net_wireless_event_t | qdf_net_wireless_event_t |
| adf_net_ipv6hdr_t        | qdf_net_ipv6hdr_t        |
| adf_net_nd_msg_t         | qdf_net_nd_msg_t         |
| adf_net_icmpv6hdr_t      | qdf_net_icmpv6hdr_t      |
| adf_csum_ipv6            | qdf_csum_ipv6            |

## 2.10 Deprecated Qualcomm driver features

The Qualcomm driver code has evolved over the last several years. Support has been introduced for several new functionalities, and most of these newly-added functionalities contain a few compilation flags. These compilation flags can be enabled or disabled through profiles. A profile is a list of valid features that must be built. As the number of flags increases, the maintenance and reading of code have also become proportionately cumbersome. Some of the features are developed only for specific radios, such as AOW or TDLS, and customers have not used such features in the production environment.

Besides the increase in the number of compilation flags, new `iwpriv` commands are also added to the wireless LAN driver. Many of the `iwpriv` commands are not used. However, these `iwpriv` commands are continued to be available. A cleanup activity to remove some of the obsolete `iwpriv` commands has been performed.

The Qualcomm drivers are currently more focused on Linux. Partial or incomplete support for other operating systems, which are ported through multiple revisions of the driver and no longer maintained, such as Windows 7, Windows XP, and FreeBSD, is available. This support is incomplete currently. While the target is to gradually remove all of these flags that are not being used, some such flags are removed in this release. A large amount of unused code is present in the Qualcomm drivers, which causes a significant impact on code maintenance. Sometimes, customer queries are also received based on the comments from such dormant or unused code. Starting with this release, an effort to start the removal of such dormant code is initiated and implemented. Remaining part of such unused code is yet to be cleared.

The following table lists all the features, flags, and code that are removed. Most of the features contained corresponding compilation flags associated with them, which are also listed in the table.

| Feature/flag                                                                                                                                                                                                                                                                                                                                                                                           | Clean up type    | Comments                                                                                                                                                                                      |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Nobeacon, get_nobeacon,                                                                                                                                                                                                                                                                                                                                                                                | iwpriv           | This feature is related to ATH_SUPPORT_AP_WDS_COMBO, through which we can create eight VAPS of A and eight WDS VAPs. This feature is not enabled in any of the profiles or related to iwprivs |
| bgscan<br>get_bgscan<br>bgscanidle<br>get_bgscanidle<br>bgscanintvl<br>get_bgscanintvl<br>coverageclass<br>get_coveragecls<br>scanvalid<br>get_scanvalid<br>regclass<br>get_regclass<br>rssilla<br>get_rssilla<br>rssillb<br>get_rssillb<br>rssillg<br>get_rssillg<br>ratella<br>get_ratella<br>rate11b<br>get_rate11b<br>rate11g<br>get_rate11g<br>radio<br>get_radio<br>ps_on_time<br>get_ps_on_time | Iwpriv/dead code | These iwprivs are listed in ieee80211_wireless.c and code is available. But all the code is under <i>notyet</i> . Unused iwpriv is removed.                                                   |

| Feature/flag                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Clean up type | Comments                                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rptcustproto<br>getrptcustproto<br>rptgputcalc<br>getrptgputcalc<br>rptdevup<br>getrptdevup<br>rptmacdev<br>getrptmacdev<br>rptmacaddr1<br>getrptmacaddr1<br>rptmacaddr2<br>getrptmacaddr2<br>rptgputmode<br>getrptgputmode<br>rpttxprotomsg<br>getrpttxprotomsg<br>rptrxprotomsg<br>getrptrxprotomsg<br>rptstatus<br>getrptstatus<br>rptassoc<br>getrptassoc<br>rptnumstas<br>getrptnumstas<br>rptstalroute<br>getrptstalroute<br>rptsta2route<br>getrptsta2route<br>rptsta3route<br>getrptsta3route<br>rptsta4route<br>getrptsta4route | Iwpriv        | As part of VOW, repeater placement feature was developed. This feature is now deprecated and all code under the flag UMAC_SUPPORT_RPT-PLACEMENT is removed. |

| Feature/flag                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Clean up type | Comments                                                                                                                                    |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| tdlsmacaddr1<br>gettdlsmacaddr1<br>tdlsmacaddr2<br>gettdlsmacaddr2<br>tdlsaction<br>gettdlsaction<br>tdlsoffchan<br>tdlsswitchtime<br>tdlstimeout<br>tdlsecchnoffst<br>tdlsoffchnmode<br>tdls<br>get_tdls<br>set_tdls_rmac<br>clr_tdls_rmac<br>tdls_qosnull<br>tdls_uapsd<br>get_tdls_uapsd<br>tdls_dtken<br>get_tdls_dtken<br>do_tdls_dc_req<br>tdls_auto<br>get_tdls_auto<br>off_timeout<br>get_off_timeout<br>tdb_timeout<br>g_tdb_timeout<br>weak_timeout<br>g_weak_timeout<br>tdls_margin<br>get_tdls_margin<br>tdls_rssi_ub<br>get_tdls_rssi_ub<br>tdls_rssi_lb<br>get_tdls_rssi_lb<br>tdls_pathSel<br>get_tdls_pathSel<br>tdls_rssi_o<br>get_tdls_rssi_o<br>pathSel_p<br>get_pathSel_p<br>tdls_tbl_query | Iwpriv        | Obsolete TDLS feature is currently removed: code under UMAC_SUPPORT_TDLS and ATH_TDLS_AUTO_CONNECT. All iwprivs under the code are removed. |

| Feature/flag                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Clean up type | Comments                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|------------------------------------------------------------------------------|
| sw_retries<br>get_sw_retries<br>aow_rtsretries<br>g_aow_rtsretries<br>aow_latency<br>get_aow_latency<br>aow_clearstats<br>get_aow_stats<br>aow_clearestats<br>get_aow_estats<br>set_aow_inter<br>get_aow_inter<br>set_aow_er<br>get_aow_er<br>set_aow_ec<br>get_aow_ec<br>set_aow_ec_ramp<br>get_aow_ec_ramp<br>set_aow_ec_fmap<br>get_aow_ec_fmap<br>set_aow_capture<br>print_aow_cptr<br>aow_force_inp<br>get_aow_channels<br>aow_playlocal<br>g_aow_playlocal<br>aow_clear_ch<br>set_aow_es<br>get_aow_es<br>set_aow_ess<br>get_aow_ess<br>aow_ess_count<br>set_aow_as<br>get_aow_as<br>set_aow_aud_ch<br>get_aow_aud_ch<br>s_aow_ctrl_cmd<br>g_aow_ctrl_cmd<br>aow_rm_ch<br>g_aow_rm_ch<br>set_aow_num_frm<br>get_aow_num_frm<br>set_aow_alt_stg<br>get_aow_alt_stg<br>set_aow_assoc<br>get_aow_assoc | iwpriv        | Obsolete ATH_SUPPORT_AOW feature and all iwprivs under the flag are removed. |

| Feature/flag                                                                                                                                                                                                                                                                                                                                                                                                      | Clean up type | Comments                                                                                                                                                              |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| s_mcastcipher<br>g_mcastcipher<br>s_p2p_opmode<br>g_p2p_opmode<br>s_wps_mode<br>s_oppss<br>s_ctwin<br>s_cancel_chan<br>s_channel2<br>s_start_hostap<br>g_conn_state<br>get_radio_idx                                                                                                                                                                                                                              | Iwpriv        | P2P is not a deprecated feature and support is not available. All the codes under flag ATHEROS_LINUX_P2P_DRIVER are removed. All iwprivs under this flag are removed. |
| get_rcpiparam<br>set_rcpi<br>set_rcpihi<br>set_rcpilo<br>set_rcpimargin<br>tdls_dtoken                                                                                                                                                                                                                                                                                                                            | Iwpriv        | CONFIG_RCPI is currently not in use.                                                                                                                                  |
| dbgcfg<br>getdbgcfg<br>numstreams<br>getnumstreams<br>streamnum<br>getstreamnum<br>nummarkers<br>getnummarkers<br>markernum<br>getmarkernum<br>markeroffsize<br>g_markeroffsize<br>markermatch<br>getmarkermatch<br>rxseqnum<br>getrxseqnum<br>timestamp<br>gettimestamp<br>dbgrestart<br>getdbgrestart<br>rxseqrshift<br>getrxseqrshift<br>rxseqmax<br>getrxseqmax<br>rxseqdrop<br>rxdropstats<br>getrxdropstats | Iwpriv        | Feature UMAC_SUPPORT_VI_DBG is currently unused. All iwprivs under this type is removed. Removal of the code in .c files is yet to be done.                           |

| Feature/flag                                                                                                                                                                                                                                                                                                                | Clean up type | Comments                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|----------------------------------------------------------------------------------------------------------------------------|
| thrput_enab<br>get_thrput_enab<br>thrput_win<br>get_thrput_win<br>get_thrput<br>PER_enab<br>get_PER_enab<br>PER_win<br>get_PER_win<br>get_prdic_PER<br>get_thrput_enab<br>thrput_win<br>get_thrput_win<br>get_thrput<br>PER_enab<br>get_PER_enab<br>PER_win<br>get_PER_win<br>get_prdic_PER<br>get_total_PER<br>thrput_enab | iwpriv        | Periodic performance statistics is deprecated and all codes and iwprivs under UMAC_SUPPORT_PERIODIC_PERFSTATS are removed. |

| Feature/flag                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Clean up type | Comments                                                                                                                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|----------------------------------------------------------------------------------------------------------------------------|
| getSmartAntenna<br>setSmartAntenna<br>ant_trainmode<br>getant_trainmode<br>ant_traintype<br>getant_traintype<br>ant_pktlen<br>getant_pktlen<br>ant_numpkts<br>getant_numpkts<br>ant_train<br>getant_train<br>ant_numitr<br>getant_numitr<br>train_threshold<br>get_tthreshold<br>retrain_interval<br>get_retimerval<br>retrain_drop<br>get_retimdrop<br>train_min_thrs<br>getrt_min_thrs<br>gp_avg_intrvl<br>g_gp_avg_intrvl<br>current_ant<br>getcurrent_ant<br>default_ant<br>getdefault_ant<br>traffic_timer<br>gtraffic_timer<br>ant_retrain<br>getant_retrain | Iwpriv        | UMAC_SUPPORT_SMARTANTENNA is deprecated; instead UNIFIED_SMARTANTENNA needs to be used.                                    |
| flowmac<br>get_flowmac                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Iwpriv        | VOW related flow control module between wireless and Ethernet is currently not in use and all code references are removed. |
| get_rb<br>rbdetect<br>get_rbdetect<br>rbto<br>get_rbto<br>rbskipthresh<br>get_rbskipthresh                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Iwpriv        | ATH_RB ( rifs burst for DA)                                                                                                |

| Feature/flag                                                                                                                                                         | Clean up type                 | Comments                                                                                                                |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| JUPITER_EMULATION<br>AR9560_EMULATION<br>AR9330_EMULATION<br>AR9485_EMULATION<br>AR9300_EMULATION_BB<br>AR9300_EMULATION<br>AR956X_EMULATION                         | Flags and code                | Emulation only code for DA radios (Osprey). Code is removed as currently there is no emulation for these.               |
| ATHR_RNWF<br>__FreeBSD__<br>MAVERICK_STA_PERF_<br>MAV_P2P_OUI<br>__NetBSD__<br>_MAVERICK_STA_<br>_WIN64<br>APPLE_DARWIN_<br>ATHR_RNWF<br>__FreeBSD__<br>REXOS<br>QNX | Unsupported operating systems | Qualcomm Atheros driver are now not supporting Windows 7, Windows XP, and FreeBSD operating systems. These are removed. |
| DEBUG_WIN_RC<br>DEBUG_RC<br>OLDCODE<br>Notyet<br>tbd<br>notdef<br>NOT_YET<br>DONTUSE<br>NOT_YET                                                                      | Other unused dead code        | This is a dead code and partially cleaned-up.                                                                           |

# 3 WLAN AP Driver Operations

---

This chapter describes the WLAN AP driver operations, such as the processing contexts, buffer abstraction, handling of transmission queues to support prioritization, and the code and data path flows.

## 3.1 Processing Contexts and Synchronization

WLAN driver execution happens in different processing context such as

- ISR context
- Softirq/tasklet context
- Process context

### 3.1.1 ISR Processing

When the WLAN device is attached (`ath_attach` function call), the driver requests an IRQ for this device and registers the `ath_isr` interrupt service routine. The driver also initializes a tasklet called `ath_tasklet` for the bottom half processing. The `ath_isr` routine first checks if the interrupt is from this device by scanning the pending interrupts from the WLAN hardware. Once the interrupt is verified to be a WLAN device interrupt, it schedules the tasklet for deferred processing through the interrupt status register. There are a few time-critical functionalities such as software beacon alert processing and UAPSD trigger processing that are done in the ISR context itself. Some of these time-critical functionalities also have the compile-time option for it to run in a tasklet context. At the end of the `ath_isr` routine, if the tasklet gets scheduled, all the WLAN device interrupts are disabled except for the interrupt corresponding to those time-critical functionalities.

### 3.1.2 Softirq/Tasklet Processing

The execution of `ath_tasklet` happens in the tasklet context. `Ath_tasklet` checks the stored interrupt status values (by `ath_isr`) and do appropriate actions. Some of the functionalities of `ath_tasklet` include

- Exception handling
- Transmit completion handling
- Receive frame handling

At the end of the tasklet, WLAN device interrupts gets enabled based on the current interrupt mask setting. The transmit frames from the networking stack through the network device hardstart

routine happens in the NETIF softirq context. All the OS timers handling are run in the TIMER softirq context.

### 3.1.3 Process Context

All driver access happens through the OS ioctl interface run in the process context.

### 3.1.4 Synchronization

Synchronizing between these processing contexts is essential since some data gets shared among these contexts. The most commonly used synchronization method used inside WLAN driver is spin\_lock. Linux provides different types of spin\_locks like spin\_lock, spin\_lock\_bh, spin\_lock\_irq and spin\_lock\_irqsave. Table 3-1 describes general guidelines on the type of spin\_lock used for synchronization among these contexts.

**Table 3-1 Synchronizing Methods**

|                  | ISR               | Softirq/Tasklets  | Process context   |
|------------------|-------------------|-------------------|-------------------|
| ISR              | -                 | spin_lock_irqsave | spin_lock_irqsave |
| Softirq/Tasklets | spin_lock_irqsave | spin_lock         | spin_lock_bh      |
| Process context  | spin_lock_irqsave | spin_lock_bh      | -                 |

Some of the other synchronization techniques used inside the driver are

- **Atomic operations:** For any primitive operations to be carried out as uninterruptable.
- **Read/write lock:** read/write locks are similar to spin\_locks, but used when read and write operations are clearly separated in the driver. Linux provides similar variants for read/write as spin\_locks and the same locking guidelines are used in deciding which type of read/write locks to be used for synchronization among these contexts.

## 3.2 Buffer Management

This section explains the buffer abstraction provided by the driver, so that frames are referenced commonly in the driver independent of how the frames are mapped to buffers from the networking stack for different OSes. Also explains the how the TX/RX frame buffers are managed inside the driver.

### 3.2.1 WBUF abstraction

#### 3.2.1.1 Wbuf

A wbuf (wireless buffer) is a platform independent object to represent a network buffer. In WLAN world, it also represents an MSDU passed down by the protocol stack. The low-level driver components treat wbuf as an opaque object defined by type wbuf\_t, and access it only through a well-defined interface. The wbuf APIs can be found in include/wbuf.h. Each platform should

implement the same set of APIs in their OS abstraction layer. Usually the wbuf is associated with or mapped to native network buffer structures. In Linux platforms it is mapped to the skb structure.

### 3.2.1.2 WBUF Type

Each WBUF object has an associated type. The WBUF types are listed in [Table 3-2](#).

**Table 3-2 WBUF Types**

| Type           | Meaning                                              |
|----------------|------------------------------------------------------|
| WBUF_TX_DATA   | Normal Tx data frames sent down from protocol stack. |
| WBUF_TX_MGMT   | Internally generated management frames               |
| WBUF_TX_BEACON | Internally generated beacon frame.                   |
| WBUF_RX        | Rx buffer that will be used Rx DMA                   |
| WBUF_TX_CTL    | Internally generated control frames                  |

### 3.2.1.3 Limitations

Ideally the wbuf abstraction should have feature parity with native network buffers. However, because wbuf abstraction must support different platforms, it only supports a common subset of APIs among Linux, NetBSD, and Windows Vista. In particular:

- Developers should not assume a wbuf is physically contiguous. A wbuf has multiple physical segments in general.
- Developers should not assume a packet is an Ethernet frame or IEEE 802.11 frame. Only after `ieee80211_encap()` can a wbuf be treated as an IEEE 802.11 MPDU.
- Developers should not assume LLC/SNAP header is in the same physical segment as WLAN header. In the same way, developers should not assume where the TCP/IP header is located.
- Developers should not assume a wbuf is associated with a large enough context area.
- A wbuf can be accessed through the public APIs in `include/wbuf.h`, however, not all APIs can be applied to every wbuf type.

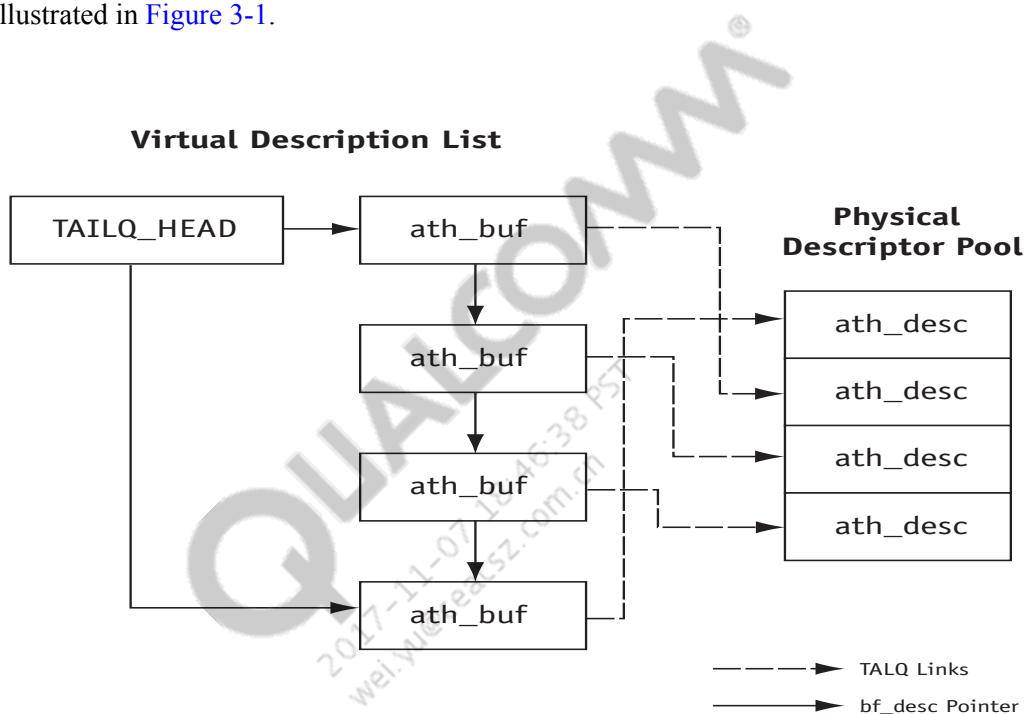
## 3.2.2 Descriptors Abstractions

The WLAN hardware use descriptors to transfer frames (which are abstracted as wbuf) between driver and the MAC hardware. The low-level driver is responsible for providing a series of descriptors to the MAC hardware. The MAC then parses the descriptors and performs the required set of data transfers. For detailed descriptions of how MAC processes descriptors, please refer to the data sheet of specific WLAN hardware chipsets.

In the WLAN driver, the descriptors are organized in pairs of physical and virtual descriptors. The driver uses the physical descriptors to communicate with the MAC. They usually have consistent DMA mapping throughout the entire life cycle of the driver, and they must not have caching issues that could prevent the processor from seeing updates made by the MAC hardware, or vice versa. A physical descriptor is represented by the `ath_desc` structure, defined in `hal/ah_desc.h`.

The physical descriptor should be treated as scarce resource due to its nature. Any control and status information accessed only by the driver should be put into the virtual descriptor, which is allocated from the normal memory pool. A virtual descriptor has a one-to-one relationship with the physical descriptor. A virtual descriptor is represented by the `ath_buf` structure, defined in `ath_dev/ath_desc.h`. The `bf_desc` pointer of the `ath_buf` structure points to the associated physical descriptor.

The virtual descriptors are managed by the `ath_dev` object as the BSD-style tail queue (TAILQ). A physical descriptor can only be accessed through its virtual counterpart. The descriptor list is illustrated in [Figure 3-1](#).



**Figure 3-1 Descriptor List**

Information stored in the `ath_buf` structure include the following:

- Reference to wbuf that needs to be transferred between driver and hardware (`bf_mpdu`)
- Physical and Virtual address of the data location the needs to be transferred
- TX descriptor flags and RX status flags for LMAC TX/RX processing

### 3.2.3 Receive Buffer Management

As part of the RX initialization, the `ATH_RXBUF` number of RX descriptors and `ath_bufs` are allocated. For each of these `ath_bufs`, a `wbuf` gets pre-allocated with the `wbuf` type as `WBUF_RX`. All these `ath_bufs` are linked as TAILQ and the `TAILQ_HEAD` is stored in `sc_rdbuf`.

On completion of a received frame from the hardware, the LMAC removes the corresponding `ath_buf` from the `sc_rdbuf` TAILQ and does Rx processing. It then removes the `WBUF` reference from the `ath_buf` and provides that `WBUF` to the UMAC and network stack for processing. A new

WBUF gets allocated, and the ath\_buf links to this new wbuf and is then added back to the sc\_rdbuf TAILQ again.

### 3.2.4 Transmit Buffer Management

As part of the TX initialization, the ATH\_TXBUF number of TX descriptors and ath\_bufs are allocated. These ath\_bufs are linked as TAILQ and the TAILQ\_HEAD is stored in sc\_txbuf.

For any frames (which are mapped to wbuf) to be transmitted from the UMAC, the LMAC removes an ath\_buf from the sc\_txbuf list. The LMAC then links the ath\_buf to the wbuf and directs the hardware to transmit the frame. Once the TX is completed for that frame, the LMAC frees the wbuf and re-queues the ath\_buf back to the sc\_txbuf list.

## 3.3 Queue Management

### 3.3.1 Hardware Queues

The WLAN hardware provides several transmit queues to support prioritization on the transmitted frames based on their access categories. The HAL module provides a set of APIs for the LMAC module to change the properties such as AIFS, CWMin, CWMax, and TXOPLimit on these queues. There is a total of HAL\_NUM\_TX\_QUEUES available (set as 10 by the HAL module). Their priority is in descending order; that is, queue9 is the highest priority queue and queue0 is the lowest priority queue.

### 3.3.2 LMAC Queuing Support

#### 3.3.2.1 ath\_txq

This is the LMAC data structure for the transmit queue corresponding to each hardware queue. In other words, this structure is a hardware queue abstraction on the LMAC layer. Since this structure has one-to-one correspondence with the hardware queue, there are a total of HAL\_NUM\_TX\_QUEUES (10) ath\_txq in the data structure ath\_softc, defined as sc\_txq[HAL\_NUM\_TX\_QUEUES]. Typically these structures are referenced as “txq” in the LMAC code.

On these 10 txqs, queues 0-3 are called DATA queues and are mapped to the four WMM access categories: background, best effort, video, and voice respectively.

txq-9 is called beacon queue (bcnq) and the beacon frames are directly queued into it.

txq-8 is called content-after-beacon (CAB) queue (cabq). This queue is used to burst the multicast frames from each VAP immediately after beacon transmissions.

The remaining txq 4-7 queues are available for any other specific needs in the WLAN driver. Some of these four available txqs are used to satisfy feature requirements; details are described in the respective feature description section.

### 3.3.2.2 Software Queues

Apart from the hardware queues, the LMAC provides software queuing support. Since the number of transmit descriptors provided by the driver is limited, having software queue support is essential for providing flow control. The LMAC provides two categories of software queues.

#### Traffic Identifier (TID) Queues

The IEEE 802.11 specification supports a maximum of 16 TIDs for each node for traffic classification. Thus the LMAC provides 16 TID queues for data traffic for each `ath_node` that gets created in the LMAC. These TID queues are represented by the `ath_atx_tid` data structure.

Data frames from the UMAC gets mapped to one of these TID queues based on their packet classification. To support queuing for management frames, one more TID queue (TID-17) has been added to the `ath_node` structure and all the management frames except beacon gets queued here.

Frames on all 17 TID queues get transmitted through hardware data queues 0-3. The mapping for these 17 TID queues for all the `ath_nodes` to the 4 hardware data queues are done through an intermediate list called access category queues. This is represented by data structure `ath_atx_ac` and typically referred to as “acq”.

- Tid queue 0 and 1 for all `ath_nodes` gets added to `acq0` list (Background access category queue).
- Tid queue 2 and 3 for all `ath_nodes` gets added to `acq1` list (Best effort access category queue).
- Tid queue 4 and 5 for all `ath_nodes` gets added to `acq3` list (Video access category queue).
- Tid queue 6 to 17 for all `ath_nodes` gets added to `acq4` list (Voice access category queue).

#### Multicast Frame Queues

For each VAP object (`avp`) the LMAC provides a software queue for the multicast frames. These queues are defined as “`av_mcastq`” in the `ath_vap` structure. Whenever any station associated to the AP is in power save mode, the multicast frames being transmitted from the corresponding VAP gets queued in this `av_mcastq` and gets transmitted out through the CAB queue.

[Figure 3-2](#) explains how the TID queues and mcast queues are mapped to the hardware queue.

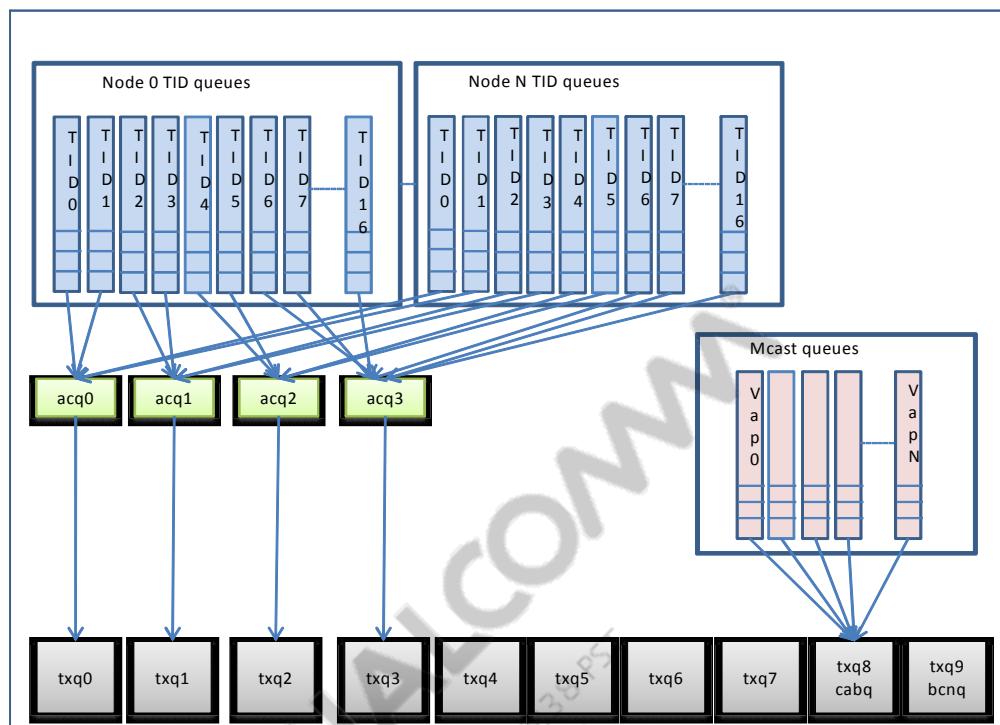
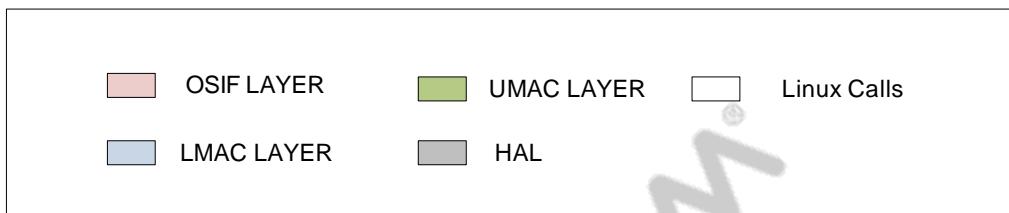


Figure 3-2 Transmit Queue Mapping

## 3.4 Code Flow

This section explains the code flow for some of the major configuration code paths and transmit and receive data paths. In these code flows, processing in each layer is represented different colors. Below is the color scheme used.



### 3.4.1 Configuration Path

The major configuration path in the WLAN driver consists of the initialization and uninitialization of the radio interface (referred to as the **wifi** network interface), and the creation and deletion of VAPs (referred to as the **ath** network interface).

#### 3.4.1.1 Driver Threads

Multiple driver threads can potentially be active at any time. These include:

- Transmit (OS context)
- Interrupt
- Deferred Interrupt Handler
- Timers
  - Beacon handling
  - Addba exchange
  - Receive Reordering
  - MLME timers (authentication/association)
- Synchronization (Lock Macros: OS-specific abstraction)

#### 3.4.1.2 Tx and Rx Initialization

Tx and Rx initialization provides the following:

- Descriptor allocation for transmit & receive
- OS-specific initialization is slightly different and handled by the OS shim
  - NDIS requires the driver to allocate receive buffers
  - Unix OS variations maintain a common buffer pool
  - For AR93xx devices, status goes into the packet buffer for the completion ring

### 3.4.1.3 Device attach

Figure 3-3 explains the radio interface initialization through the PCI bus interface (initialization through other bus interface is not discussed here). As soon the WLAN driver gets loaded, the radio interface initialization starts through `ath_pci_probe`. This device attach gets called for each radio interface in the system.

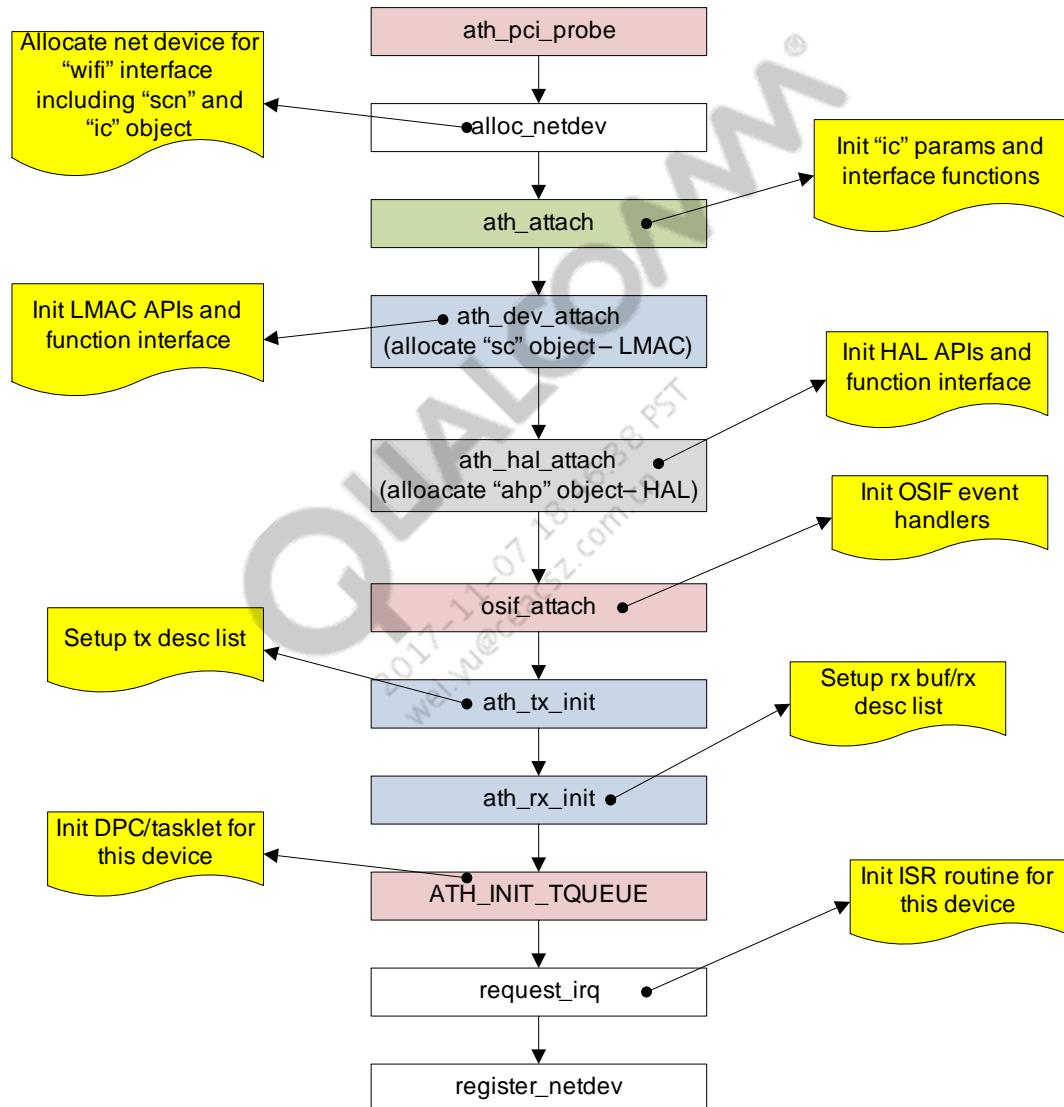
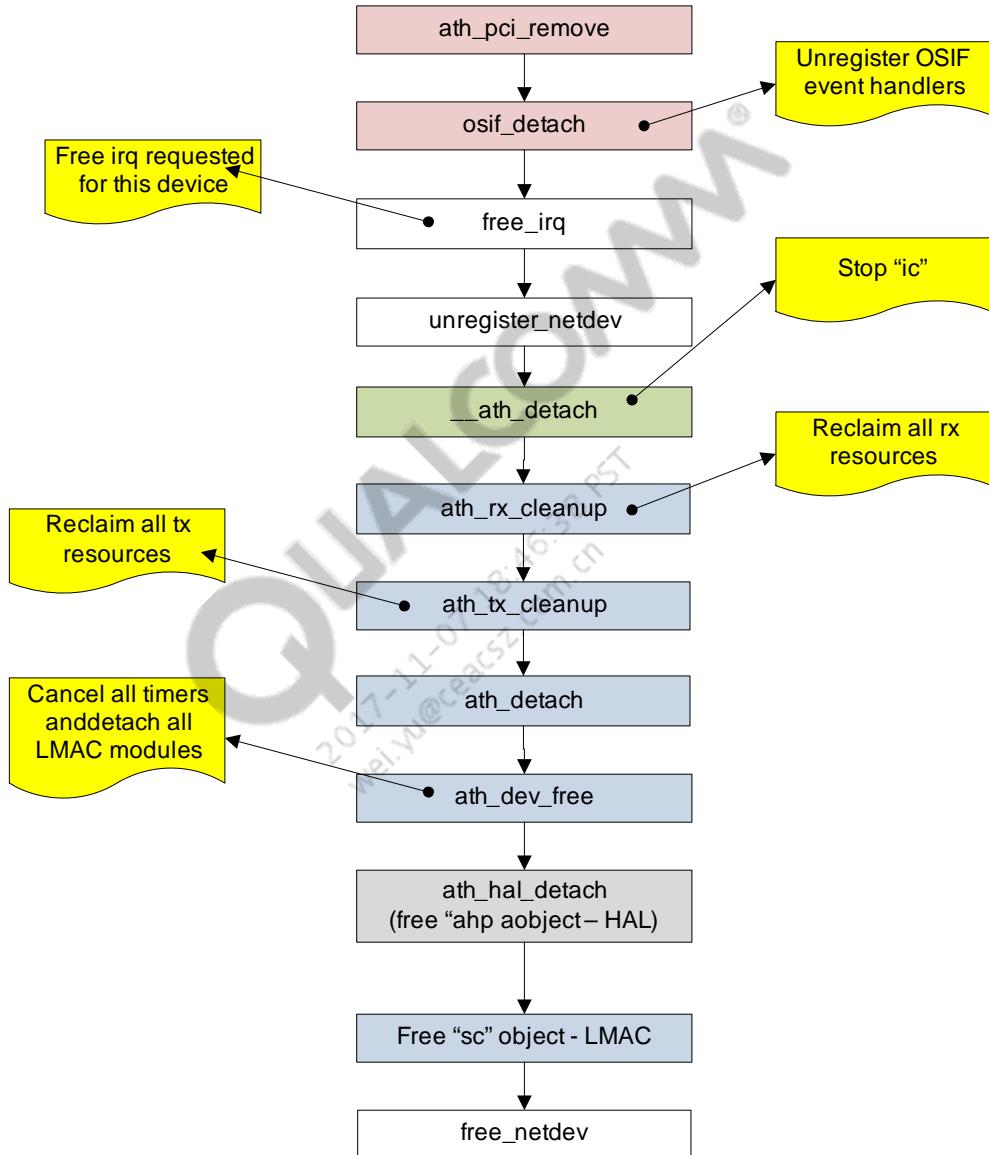


Figure 3-3 Radio Interface Initialization (PCI Mode)

### 3.4.1.4 Device detach

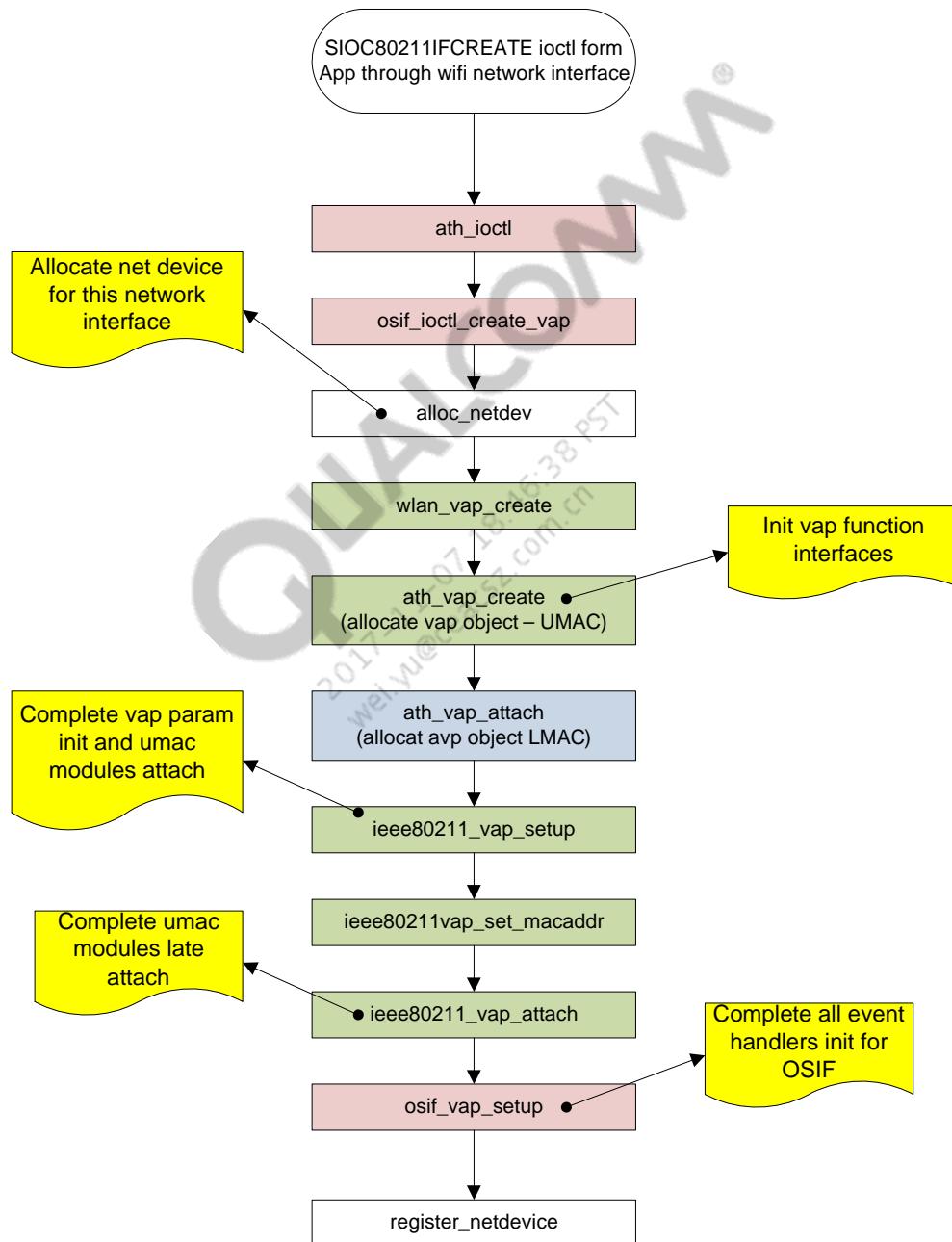
The radio interface gets removed when the driver is unloaded and the interface is removed through the `ath_pci_remove` function, which is registered as a remove routine in the `pci_driver` structure.



### 3.4.1.5 VAP create

The VAP network interface gets created through the “wifi” radio network interface ioctl SIOC80211IFCREATE. This “wifi” interface is the parent device for this VAP interface.

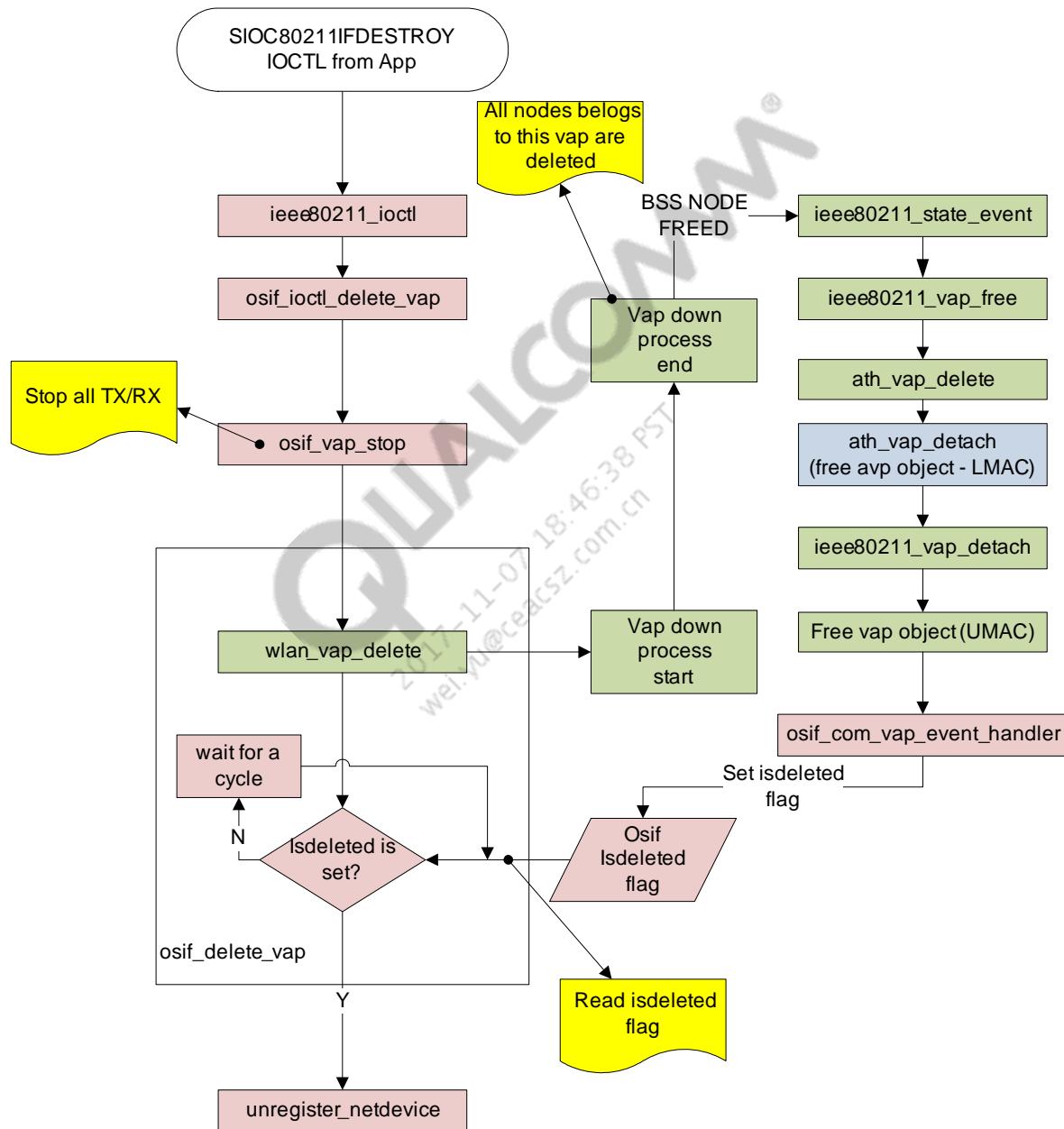
[Figure 3-4](#) explains the code flow for the VAP create. Each VAP maps to corresponding “ath” network interface.



**Figure 3-4 VAP Create Codeflow**

### 3.4.1.6 VAP delete

A VAP delete is triggered from the application through the VAP network interface ioctl SIOC80211IFDESTROY.



**Figure 3-5 VAP Delete Codeflow**

### 3.4.1.7 VAP start

A VAP network interface starts through the network device open callback function, which is registered during a VAP create.

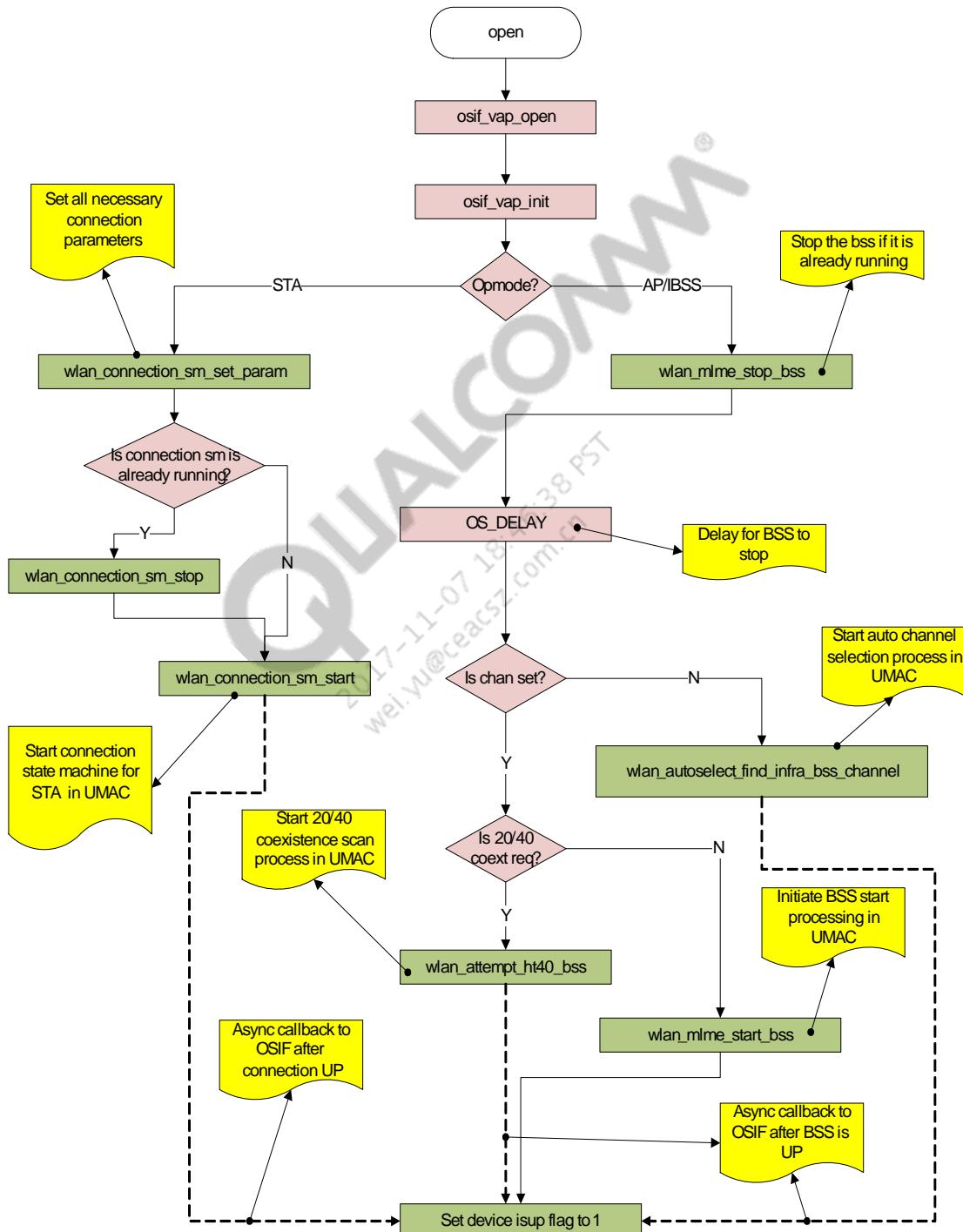
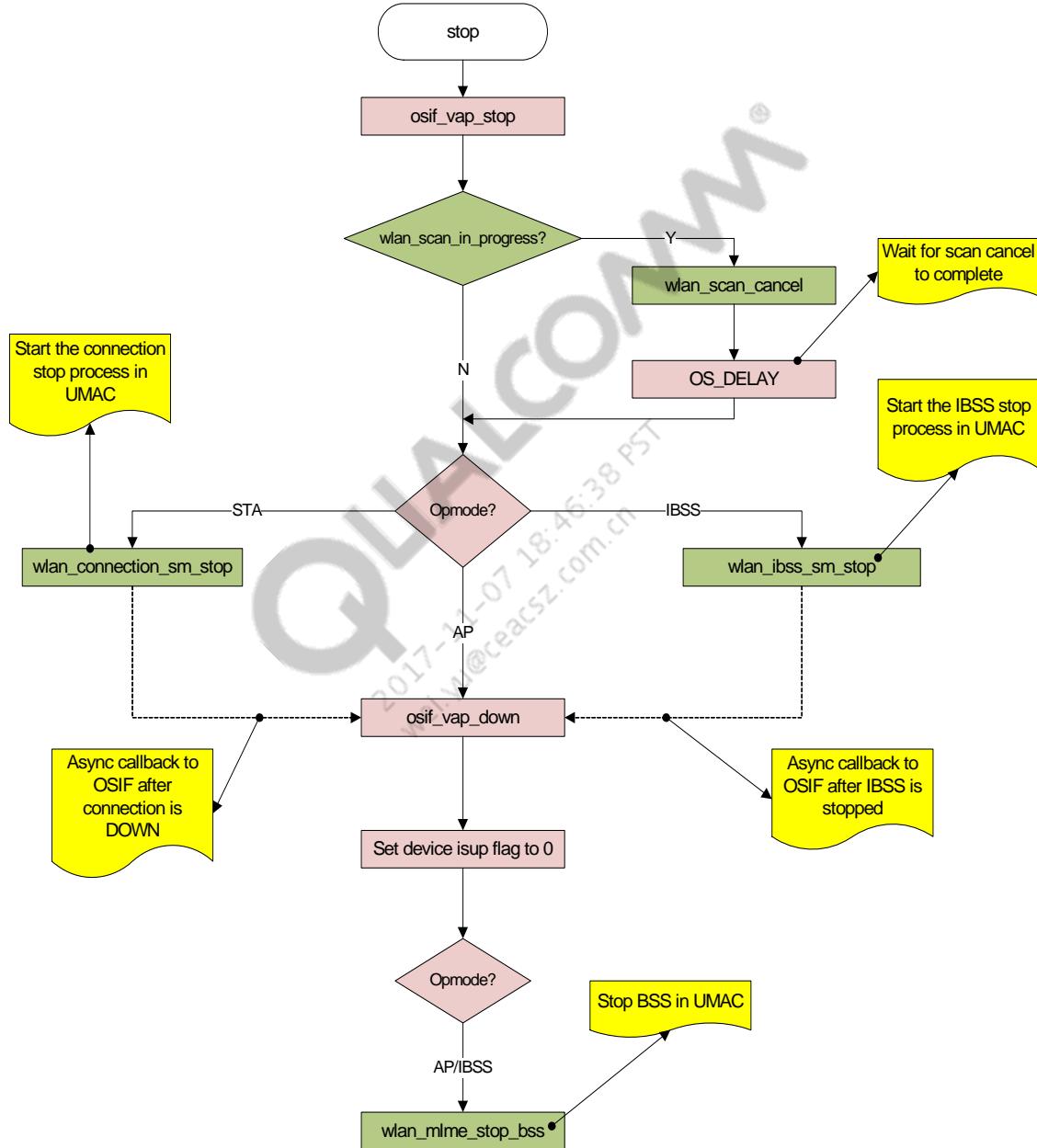


Figure 3-6 VAP Start Codeflow

### 3.4.1.8 VAP stop

A VAP network interface is stopped through network device stop callback function, which is registered during a VAP create.



**Figure 3-7 VAP Stop Codeflow**

## 3.4.2 Data Path

### 3.4.2.1 Transmit flow

Transmit flow occurs in two different execution paths. The WLAN driver gets the data packet from the network stack and after driver processing, the packets either gets queued in the software TID queue or gets transmitted out to the hardware (assuming there is no pending transmit packet in the hardware queue). On packet transmit completion, the hardware asserts the transmit completion interrupt to the driver, and then the driver schedules the tx\_tasklet for transmit completion processing. As part of the tx\_tasklet, the driver releases the transmitted packet and runs the TID scheduler to queue the packets from the software TID queue to the hardware queue.

The data transmit path originates in the OS shim layer, and has the following characteristics:

- Calls wlan\_vap\_send (umac/txrx/ieee80211\_output.c) to start the process
- Win7/Vista hands off 802.11 packet, Others hand off 802.3
- Node lookup and authorization
- UMAC hands packet to **ath\_tx\_send (umac/if\_lmac/if\_ath.c)**, prepares packet
  - Encapsulate if needed, crypto headers, tid, crypto
- Packet comes down to **ath\_tx\_start\_dma (lmac/ath\_dev/ath\_xmit.c)**
  - Physical mapping to descriptor is completed
  - Descriptors are allocated and created
  - If hardware queue has fewer threshold packets, queue single packet to hardware, or else packet is put on a “tid” queue
- Aggregate is created when handling interrupt
  - Interrupt is specific to a queue
  - When tx interrupt happens, the driver scans nodes that have packets to transmit
  - One of these nodes is picked in “round robin” fashion
  - Aggregate is created and queued to hardware

### 3.4.2.2 Transmit Interrupt Handling

The transmit interrupt is generated per queue. On interrupt completion, **ath\_tx\_edma\_tasklet()** is called. Rate information is then collected (**ath\_rate\_tx\_complete\_11n** is called):

- For single packets, retry if software retry is enabled
- For aggregates, **ath\_tx\_complete\_aggr\_rifs()** is called to process block ack and returns any incompletes to the head of the transmit queue for <node,tid>
- If the maximum number of retries is exceeded, complete the packet to the OS; the OS shim calls for OS-specific completion
- Schedule the next transmit by calling **ath\_txq\_schedule()** (**ath\_xmit\_ht.c**)
  - Look through nodes with packets to send and pick one (round robin)
  - If addba is complete, create an aggregate (previously failed packets will be part of the new aggregate)
  - Lookup rate control and pick rate series
  - Queue packet/aggregate to hardware for transmit

### 3.4.2.3 Transmit Path Management

The code in **ieee80211\_send\_mgmt** (**wlan/umac/mlme/ieee80211\_mgmt.c**) is used for the following tasks:

- probe request
- probe response
- beacon
- action frames
- null data packets

The packet is prepared similar to data packets. Rate control is forced to use minimum data rate, and the voice queue is used for all management packets.

The transmit path management interfaces into ath layer via **ath\_tx\_mgt\_send()**. When the management task is complete, the code flow path is the same as data packets.

### 3.4.2.4 Interrupt Path

An interrupt is attached by the OS shim during driver initialization. The sequence is as follows:

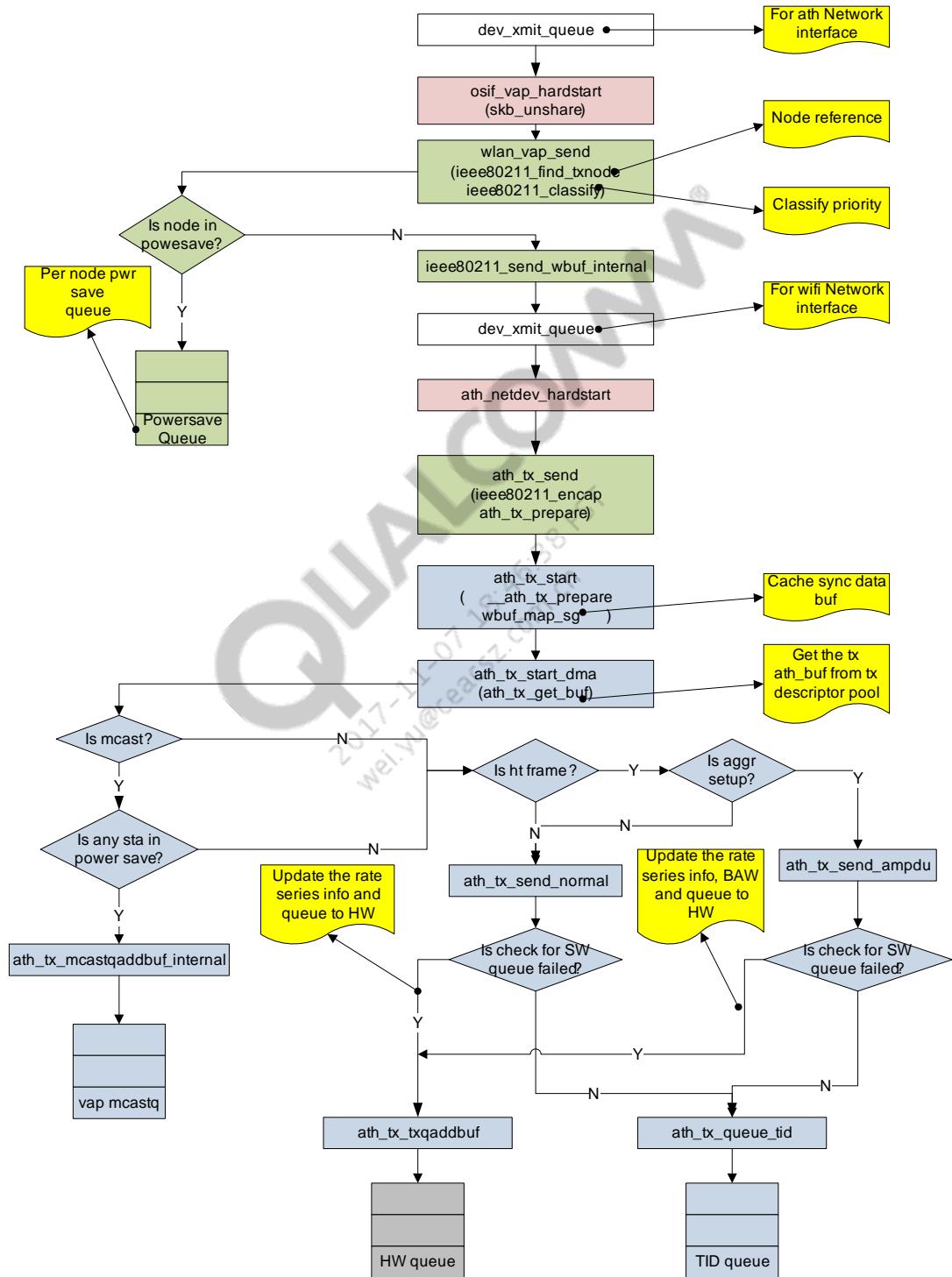
- Entry point `sc->sc_ops->isr()` is mapped to `ath_intr`
- `ath_intr` (`lmac/ath_dev/ath_main.c`) calls into the HAL (`ah_isInterruptPending`) to get pending interrupts
- `ah_isInterruptPending` maps to a device specific function – the AR93xx HAL uses `ar9300IsInterruptPending()`
- HAL translates bits from chip specific to common mapping:
  - `HAL_INT_SWBA`
  - `HAL_INT_RX`
  - `HAL_INT_TX`
  - ...
- `ath_intr()` returns a boolean value indicating interrupt ownership
- `ath_handle_intr()` is scheduled by OS.
  - The OS shim links up to
    - Deferred procedure call (DPC) interface in Windows
    - handle\_interrupt interface in MacOS
    - Immediate call in netBSD
  - Calls specific tasklet for handling each interrupt
    - Transmit interrupt handling for AR938x/AR939x ends up in `ath_tx_edma_tasklet()` in file `ath_edma_xmit.c`
    - Receive interrupt handling similarly ends up in `ath_rx_edma_tasklet()` in file `ath_edma_recv.c`
  - Other interrupt causes are handled similarly
    - Beacon
    - GPIO
    - Errors such as overrun/underrun/fatal-chip-error

### 3.4.2.5 AR93xx Transmit

For AR93xx devices, status completion is reported in a common completion ring. Each hardware descriptor points to up to four physical memory fragments. Multiple descriptors can be chained.

- TX Descriptor Pointer (TXDP) FIFO
  - 8 deep
  - 1 FIFO per queue
  - 2 (at most) for aggregates

## Data Transmit flow



**Figure 3-8 Data Transmit Flow**

### 3.4.2.6 Data Transmit Completion Flow

Figure 3-9 explains the TX completion interrupt handling and how the tx scheduling is done as part of transmit completion handling.

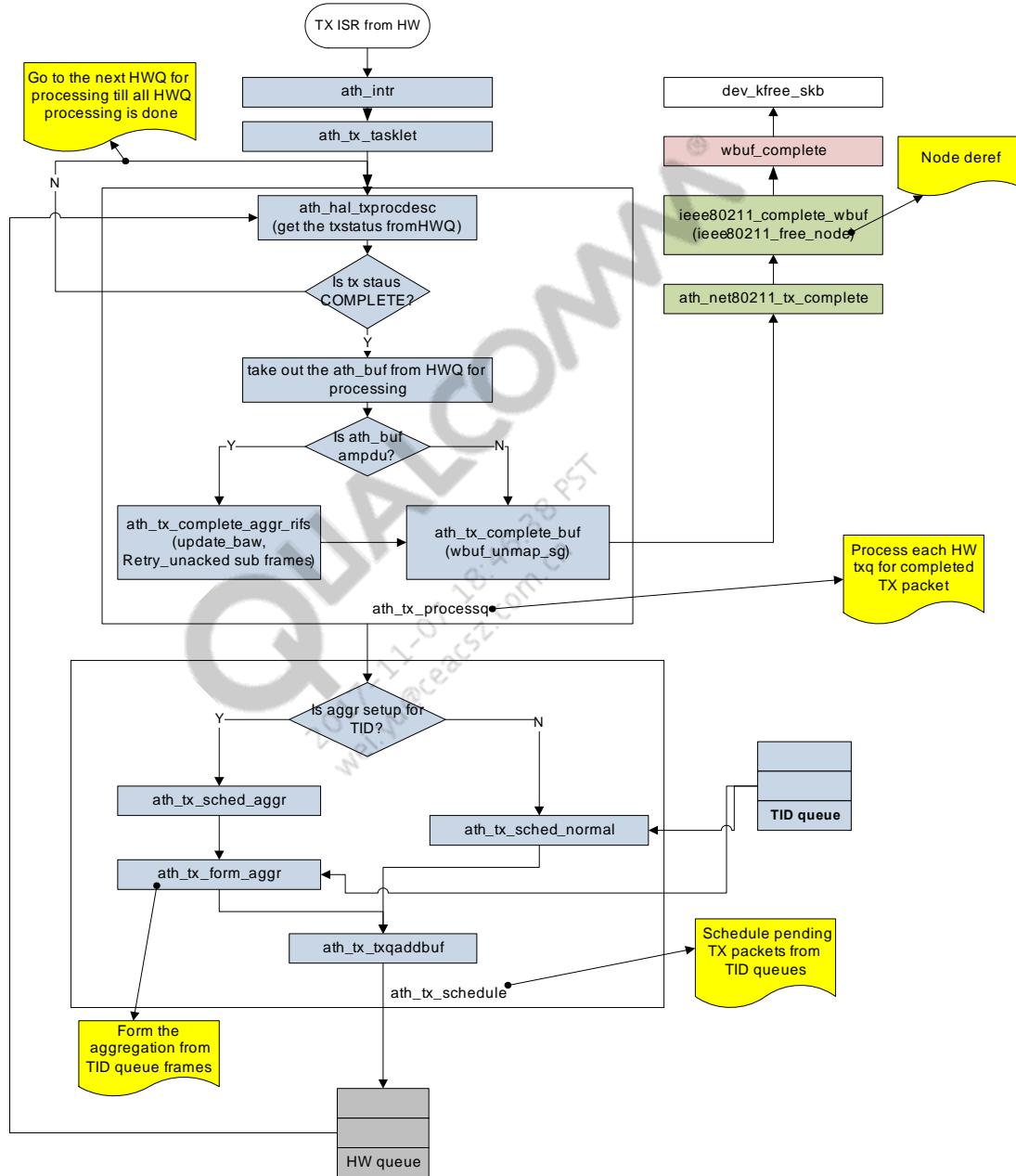
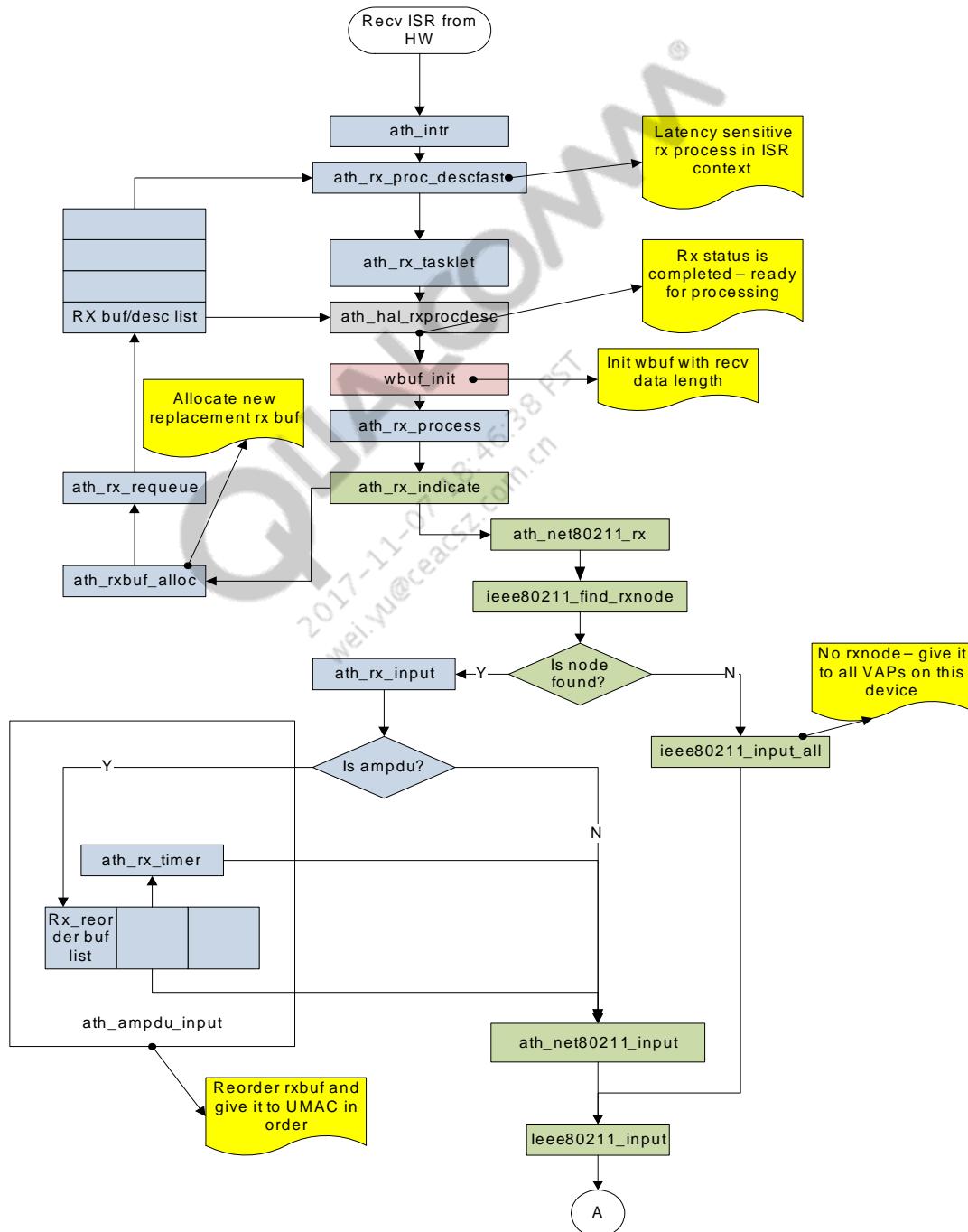


Figure 3-9 Data Transmit Completion Codeflow

### 3.4.2.7 Receive flow

The receive frame from hardware gets indicated to the driver through HAL\_INT\_RX. There are some receive processing which are latency sensitive (like UAPSD trigger processing) and in those cases, processing is done in the ISR context. These rx processing should be kept to a minimum. The remaining majority of the rx processing is done in a tasklet context and handled by ath\_rx\_tasklet. The detailed rx frame processing code flow is explained in [Figure 3-10](#) and [Figure 3-11](#).



**Figure 3-10 Receive Codeflow (Chart 1)**

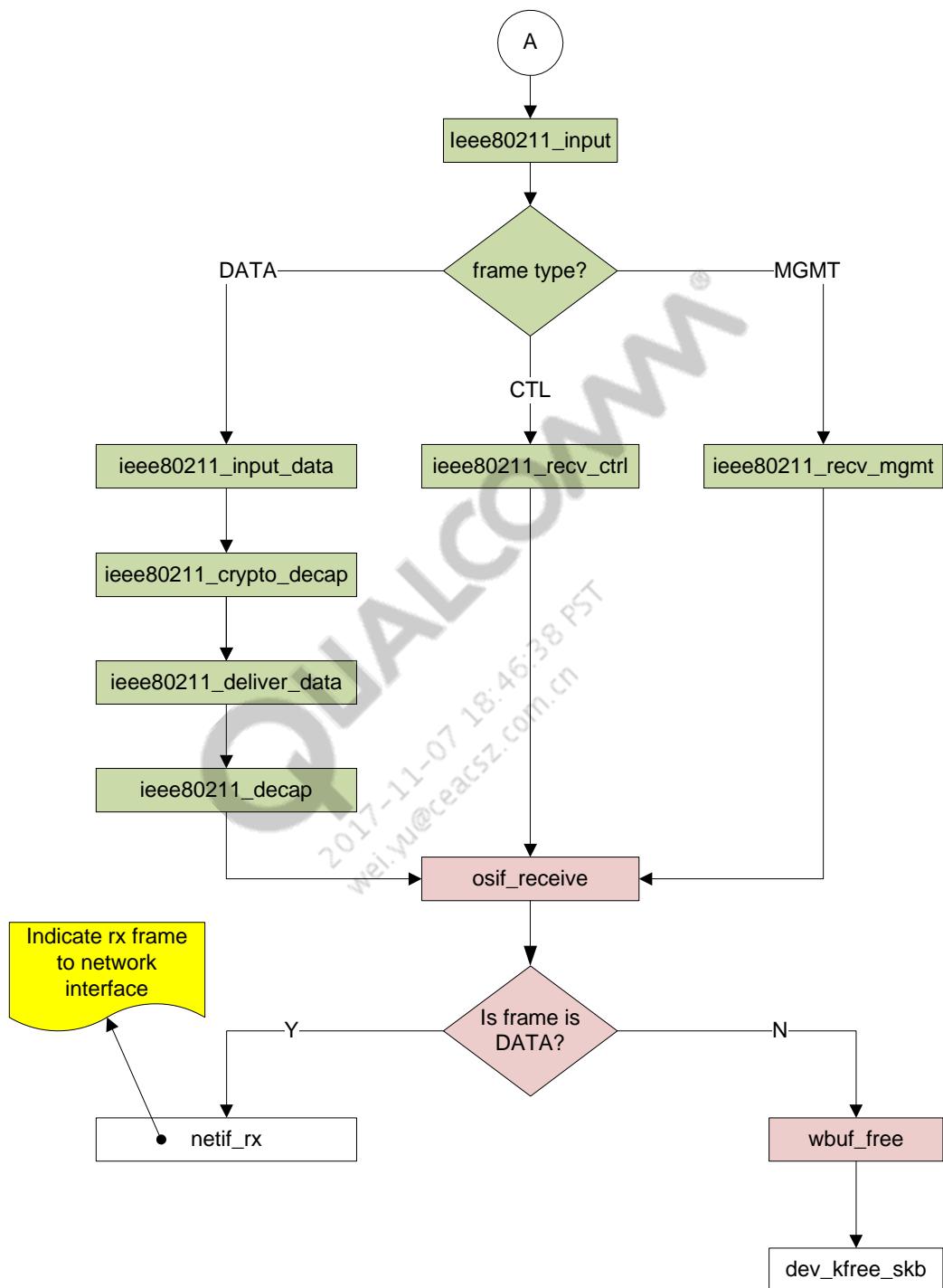


Figure 3-11 Receive Codeflow (Chart 2)

### 3.4.2.8 Single Data Packet Receive

During a single data packet receive task, the packet is processed by **ieee80211\_input\_data()**. It is parsed and sorted by mode; STA/HOSTAP/IBSS handle broadcasts differently. Other events include:

- Crypto DeCap and DeMic; gives software the opportunity to process crypto and demic
- AMSDU processing is done if needed (**ieee80211\_amsdu\_input()**)
- Calls **ieee80211\_deliver\_data** to translate to 802.3 if needed
- In HostAP mode, packet can be forwarded out to the wifi medium if the destination node is associated, or to the bridge interface above
- In STA mode, packet is handed up to the stack

### 3.4.2.9 Aggregate Receive

During an aggregate receive, the packet is handed back to the ath layer for AMPDU processing. **ath\_ampdu\_input()** (in **lmac/ath\_dev/ath\_recv\_ht.c**) gets the packet.

- If qosdata, tid is extracted
- Relevant Rx block ACK window (rxbaw) structure for the tid is pulled out of the node.
- Seq# is mapped into the baw, all in sequence frames are handed to upper layer (umac) at this point.
- A timer is maintained to manage holes – if timer fires, all packets are completed and the window is advanced.

Packets that are completed are processed through same path as single data receive as described earlier.

### 3.4.2.10 AR93xx Receive Handling

The AR93xx chips support high and low priority receive “rings”. High priority is used for UAPSD. Low priority is normal TID traffic.

**ath\_rx\_edma\_init()** (in **wlan/lmac/ath\_dev/ath\_edma\_recv.c**) is called to init receive a fixed buffer size, initialize the FIFO software data structures, and allocate buffers that call OS-specific allocation functions in **ath\_osdep.c**.

When ready to start receiving, **th\_edma\_startrecv()** is called to post allocated buffers, enable the receive FIFO in hardware, set filters, set the MAC address, and start the DMA engine.

### 3.4.2.11 Receive Interrupt

When a receive interrupt is raised, a DPC function calls **ath\_rx\_edma\_tasklet()**, which handles receives for both high and low priority queues. The function:

- Polls the first posted buffer for “done” bit
- If done, checks status, pulls good packets out, and processes them to completion

- Posts bad packets back to the hardware queue for reuse

Packets pulled off the completion ring are passed into **ath\_rx\_process** (in the **ath\_recv.c** file). This is a common function for legacy hardware. Status bits are processed, and packets with status are passed to **ath\_rx\_indicate** in the **ath\_osdep** file.

### 3.4.2.12 ath\_rx\_indicate() for MacOS

For this function, a packet is indicated to the upper layer (UMAC) by calling entry point **sc\_ieee\_ops→rx\_indicate()**. A new buffer is allocated for the hardware, and then sent to the hardware-ready queue.

### 3.4.2.13 Other Receive Events

**sc\_ieee\_ops→rx\_indicate()** maps to **ath\_net80211\_rx()** in **if\_ath.c** (in **umac/if\_lmac**), which in turn:

- can “show” the packet to a monitor mode interface
- locates a “node” associated with this packet
- calls **sc\_ops→rx\_proc\_frame** in ath layer for “ampdu” nodes
- for single packets, packet is processed further in **ieee80211\_input()** (in **umac/txrx/ieee80211\_input.c** file)
  - data packets go to **ieee80211\_input\_data()**
  - management packets into **ieee80211\_recv\_mgt()**

## 3.4.3 Data Path — Partial Offload

In the new generation of 802.11ac wireless chips, such as the QCA988x/989x and QCA999x/998x, much of the data-path functionality is offloaded to the processor on the chip itself (target processor). The processor on the AP SOC platform such as the AP135 runs the small host data-path code. On AP platforms, the “low latency” code path is used. Communication between the host and the target is through a DMA entity called Copy Engine. The “Copy Engine” has 8 pipes. Each pipe in the Copy Engine, can be configured to talk either from host to target, or from target to host

All notifications from target to host are in the form of interrupts. The interrupt handler

1. checks for the exception /error interruptsDat
2. turns off interrupts
3. schedules a delayed processing routine (for example, a tasklet for Linux)

In the partial offload architecture, the division of functionality between the host and the target can be broadly classified as the following:

### Host Tx

- DMA map the data buffer, for the packet or list of packets received from stack

- Set up meta-data required for target
  - Set up the copy engine to do a transfer of the meta-data & part of the data packet.
  - Do Tx completion, which can happen in 2 steps:
    - Tx descriptor download complete
    - Tx packet complete
- Only when both parts are completed, the packet is freed.

### Target Tx

- Packet classification into WMM access classes.
- Aggregation per TID, aggregation setup & teardown.
- Rate lookup.
- Indicate packet completion via HTT messages.

### Host Rx

- Post Rx buffers into which MAC hardware can DMA.
- Discard the packets or reorder & deliver the packets up to the stack (depending on HTT message indication)

### Target Rx

- Block Ack Window Management and sending of Block Ack.
- Posting HTT Rx indication messages to the host

The header encap/decap and the encryption/decryption (if security is enabled) is done by the MAC hardware.

[Figure 3-12](#) captures the sub-modules exercised in the data-path:

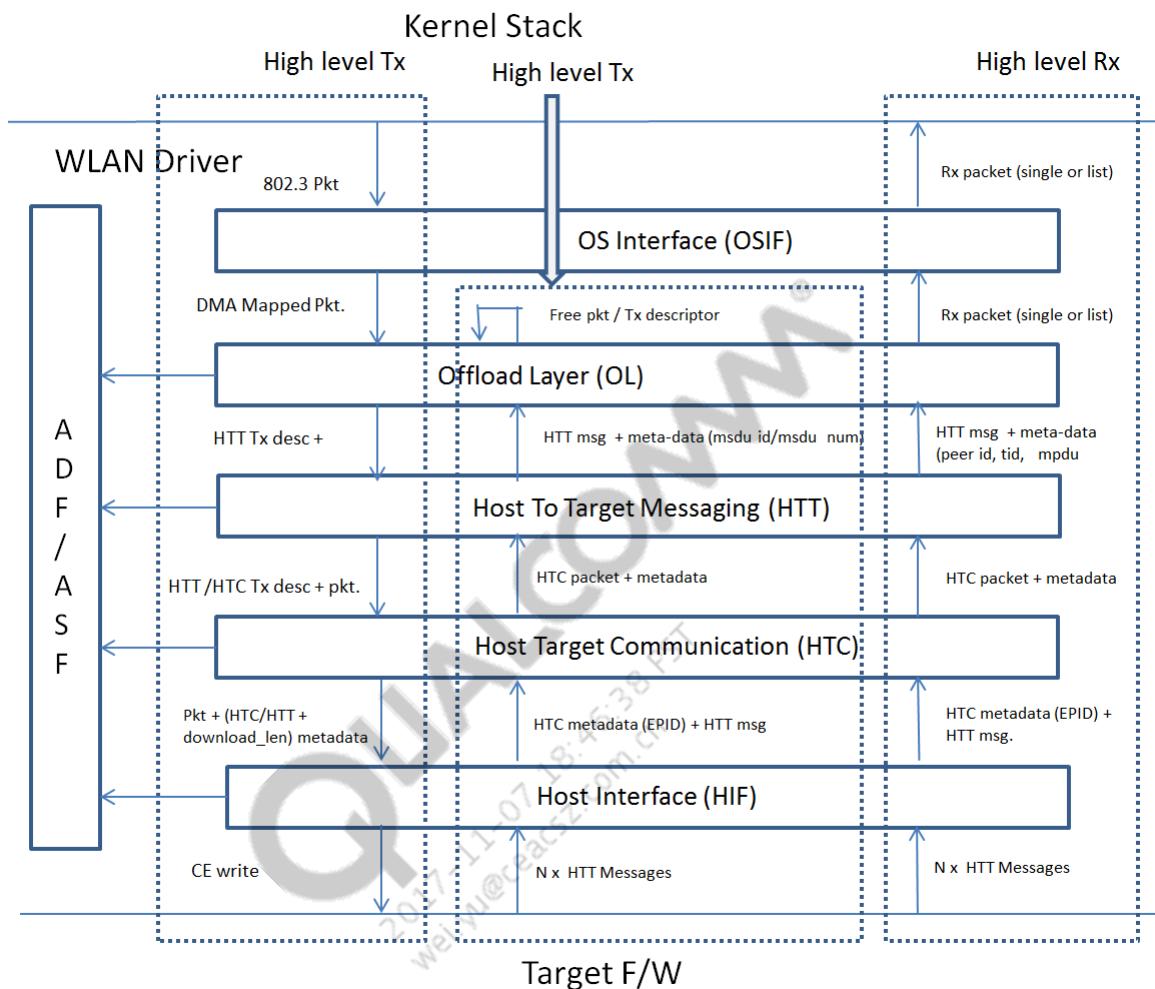


Figure 3-12 TxRx Architectural Overview

### 3.4.3.1 Tx Processing

For partial offload path, the Tx processing can be divided into two parts:

- Transmission of the packet
- Completion
- indication of the transmitted packet

The WLAN driver can receive the following types of packets from the stack (depending on configuration):

- Plain Ethernet packet (with or without a VLAN header)
- A native Wi-Fi (802.11) packet (with or without a VLAN header)
- A raw 802.11 packet

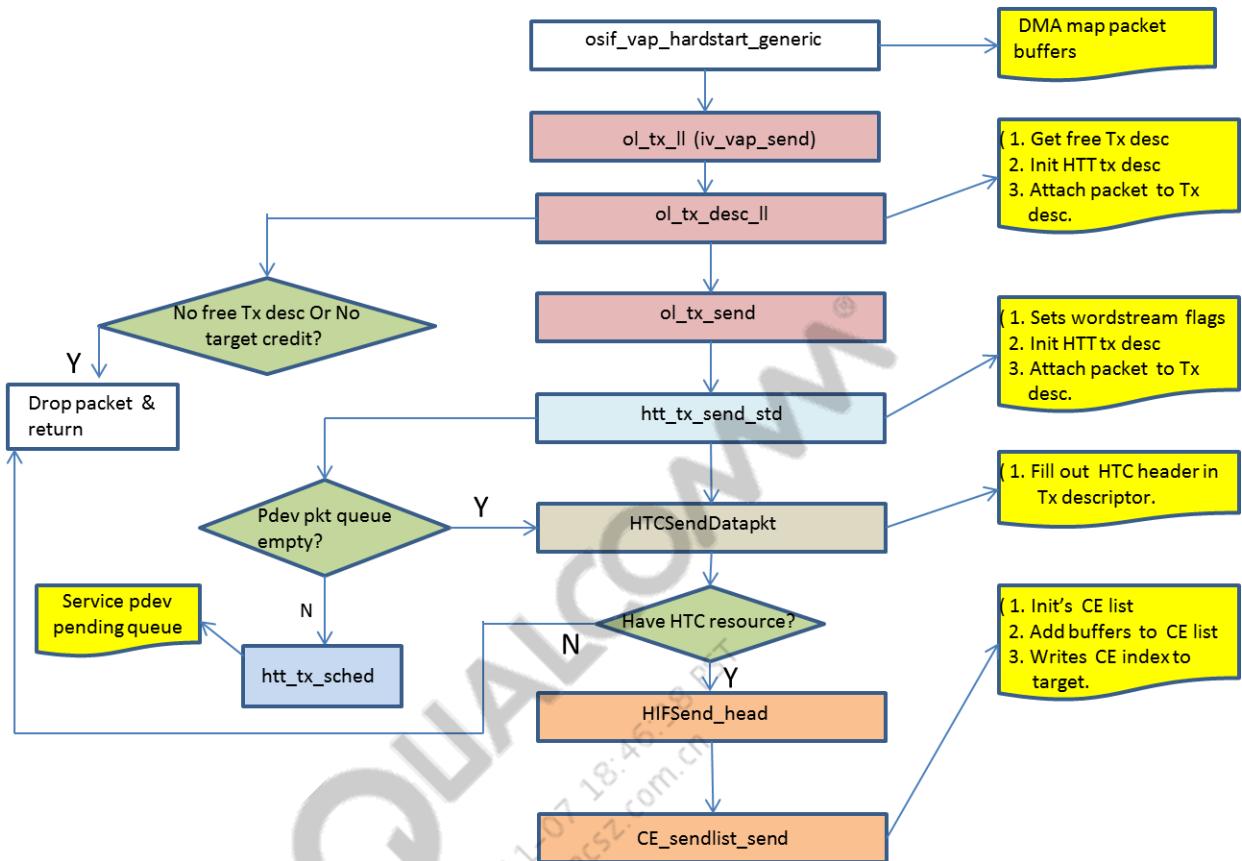
Following the salient features of the Tx path processing:

## Transmission of packet

- Packet enters the driver through a OS shim specific entry point. DMA mapping of the buffer/list of buffers is done at this point. The packet is handed off to the common ‘Offload’ (OL) layer for further processing.
- The OL layer entry point is **ol\_tx\_ll()**
- **ol\_tx\_ll()**
  - allocates the OL level Tx descriptor
  - stores the fragment meta-data inside the control block of the packet (struct cb {} for linux)
  - prepares the HTT descriptor embedded inside the OL Tx descriptor
  - fills out the scatter gather list inside the HTT descriptor
  - Calls **ol\_tx\_send()**
- **ol\_tx\_send()** passes the packet to the HTT layer.
- **htt\_tx\_send\_std()** adds HTT/HTC fragment to the qdf\_nbuf meta-data, sets endianness per fragment, allocates and prepares a HTC (software only, internal) packet and calls HTC layer.
- **HTCSendDataPkt()** checks for available HIF resources, and drops the packet if there is no sufficient resources. It fills out the HTC descriptor & passes the packet to HIF layer.
- **HIFSend\_head()** adds the fragments in qdf\_nbuf\_cb meta-data (fragments passed by stack + HTT/HTC fragment) to Copy Engine (CE) send-list and invokes the CE.
- **CE\_sendlist\_send()** posts the buffers to target by writing the copy engine ring index to the corresponding CE register on the target. The number of CE slots used up per packet is equal to the number of fragments (= DMA mapped addresses) in the packet. Since linux sends 1 packet at a time, from the stack, no. of CE slots used per packet is 2.
- **CE 4** is used for the host to target of HTT + HTC descriptor and a portion of the data packet.

**NOTE** For a given packet, the driver downloads 24 bytes of HTT/HTC header + a certain portion of the data-packet to the target. Even though the data-packet is DMA’ed by the MAC h/w directly from host memory, this partial download is required so that the target can parse the required packet headers for packet classification. This partial download len includes the L2 headers + 2 bytes of L3 headers (for IPv4 TOS / IPV6 Flow label) information. The download length is set at driver initialization time and has the following values:

Ethernet packet → 28 bytes  
Native 802.11 packet → 44 bytes  
Raw 802.11 packet → 50 bytes



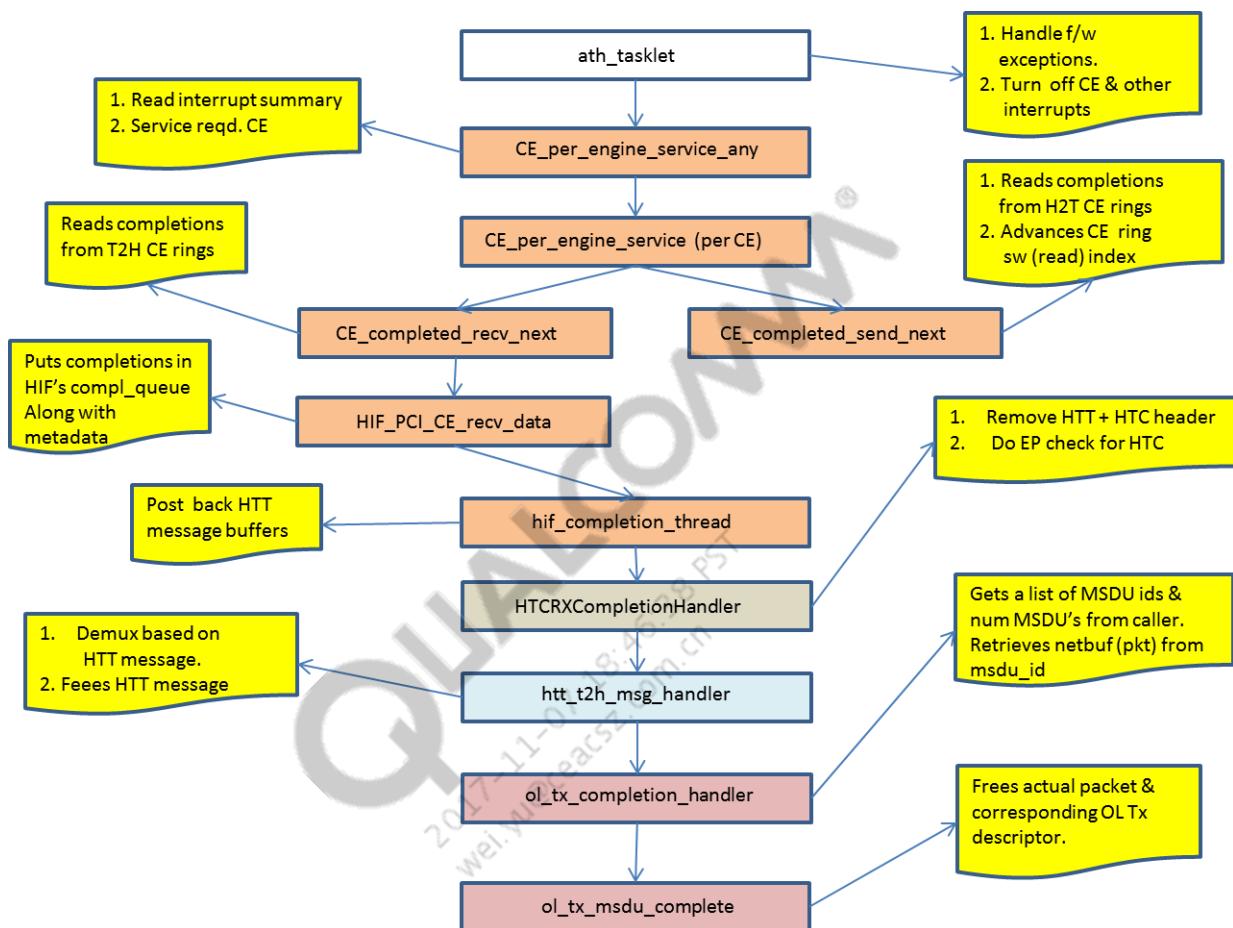
**Figure 3-13 Transmit Path Flow**

### Completion of Transmit Packet

Tx completion notification has two parts:

1. Download completion of the descriptor to the target.
  - This is just an interrupt from the target to the host to indicate that the descriptor is complete.
  - This can be used to free the Copy Engine indexes, but not the actual packet meta-data.
  - For CPU efficiency, this interrupt for CE 4 has been disabled. Instead, this copy engine hardware index is polled to find the number of descriptors for which download is complete.
2. Completion of the actual packet transmission.
  - This comes as a HTT message from the target to the host.
  - Notification is in the form of target to host interrupt per Tx PPDU on CE 1, which is used for target to host HTT messages.
  - The HTT message has the IDs of all packets that were completed. Using these IDs, the actual packet and all the meta-data such as HTC packet, OL Tx descriptor, and so on, are freed.

- Buffers for HTT messages per copy engine are freed after processing the message is complete. They are re-allocated and posted back in HTT message handler, before this function returns.



**Figure 3-14 Transmit Completion Flow**

### 3.4.3.2 Rx Processing

Receive consists mainly of two major parts:

- Processing of Rx indications from the target to the host, which comes in the form of HTT messages
- Allocation and posting of Rx buffers, into which the actual data is DMAed by MAC hardware.

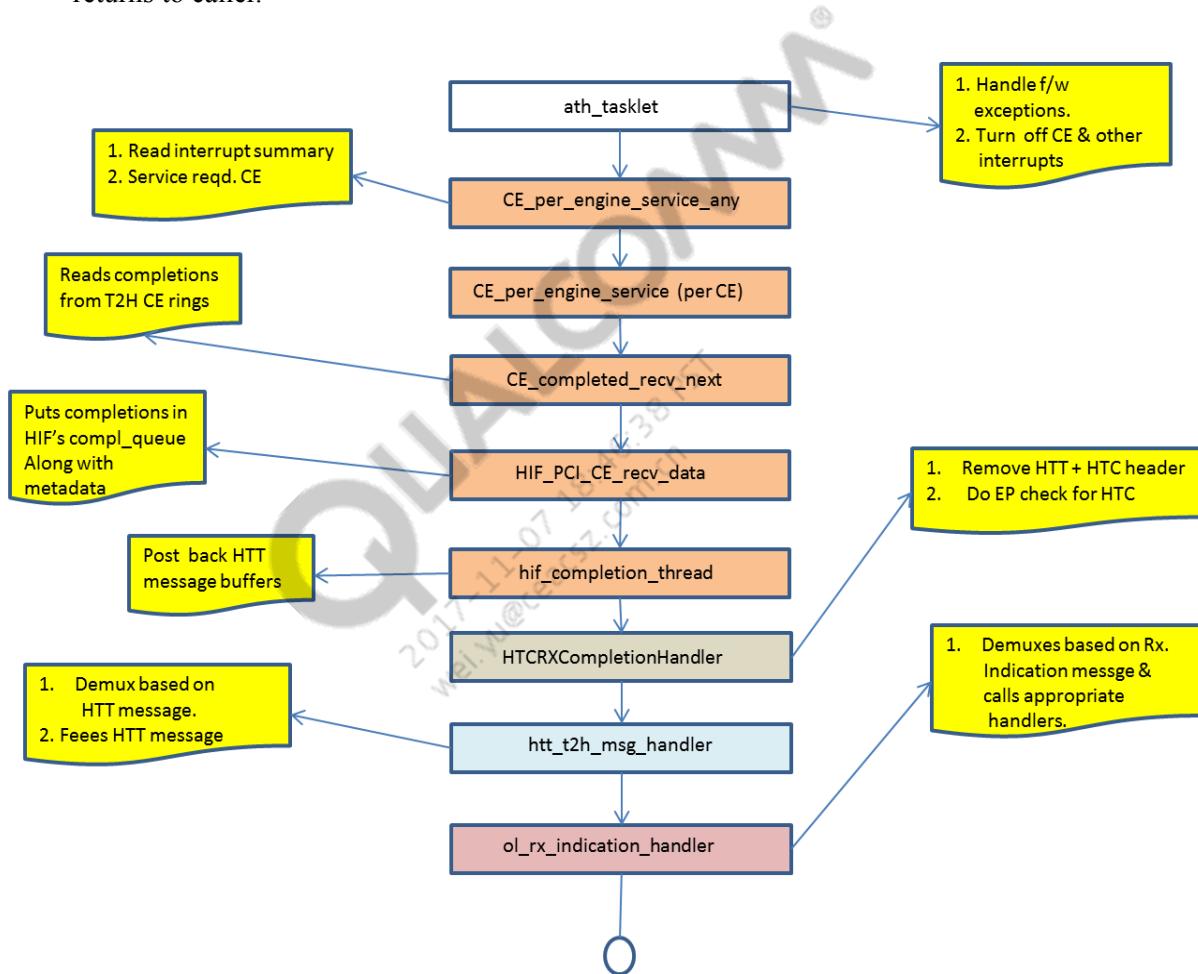
#### Processing of Rx Indications

- Rx Indications from the target to the host are in the form of a per PPDU interrupt.
- Rx Indications are conveyed from target to host using HTT messages, and hence it uses the target to host dedicated copy engine 1.
- CE handlers dequeues the HTT message buffers from the TID queue, which hands over the HTT message to HIF, to HTC, and finally onto HTT.

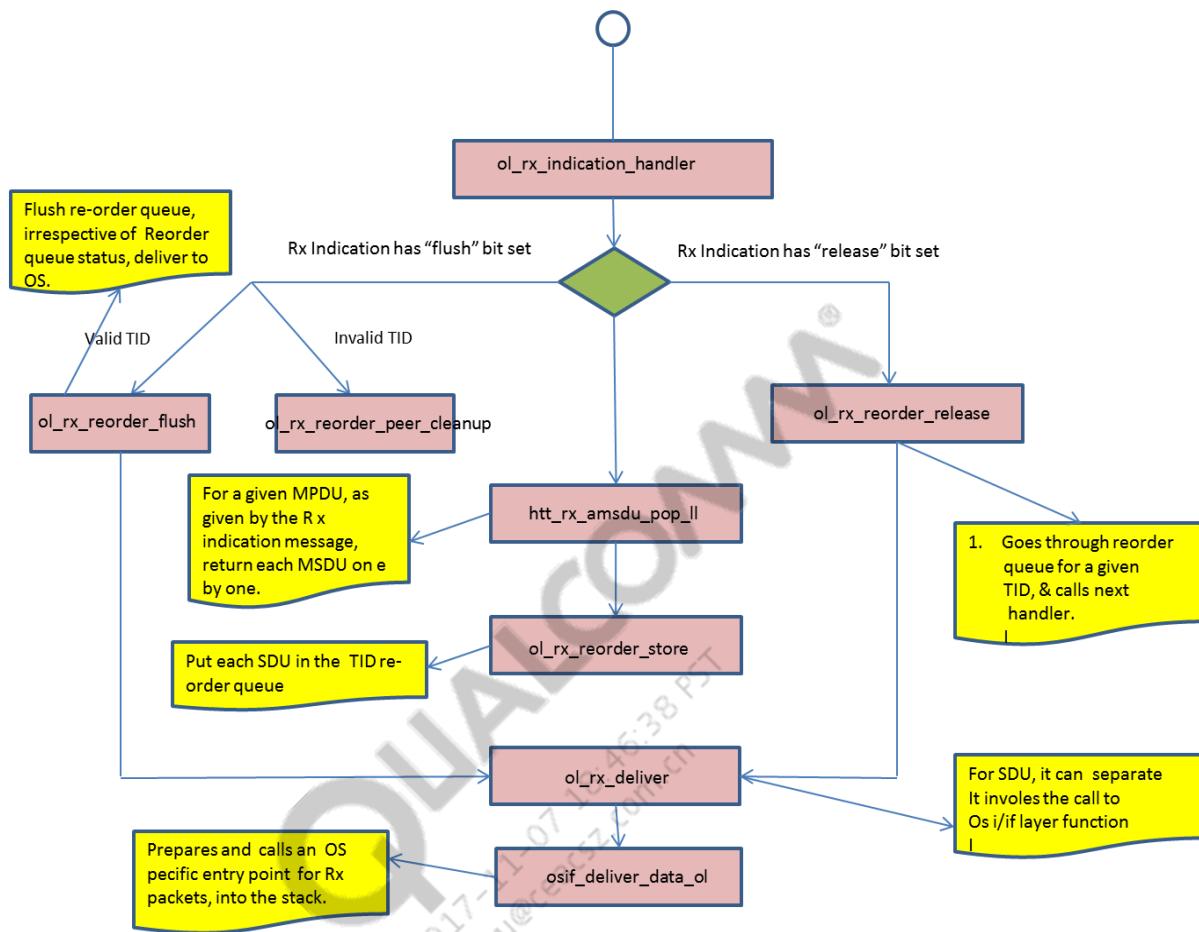
- HTT message handler (`htt_t2h_message_handler()`) de-multiplexes message and calls the OL handler `ol_rx_indication_handler()`.
- `ol_rx_indication_handler()` looks at the Rx descriptor that precedes the actual Rx buffer, pulls out the MSDUs from the MPDUs, and delivers each of the MSDUs to the stack.

### Allocation and posting of Rx buffers

Rx buffers are posted back to hardware inside `ol_rx_indication_handler()`, just before the function returns to caller.



**Figure 3-15 Rx Processing Flow (Part 1)**



**Figure 3-16 Rx Processing Flow (Part 2)**

### 2.5.3.3 Data Path Data Structures

The following subsections list some of the important structures for the offload layer data-path.

#### **ol\_txrx\_pdev\_t: Object corresponding to TxRx physical device**

This represents the transmit and receive characteristics in the offload mode of the physical device associated with the WLAN driver. An OL physical device stands for a radio interface (such as *wifi0*, *wifi1*, and so on). This structure maintains the following important information:

- Opaque OS device handle
- Handle to lower layer physical device such as HTT
- Frame format for this particular physical device (Ethernet, native Wi-Fi, raw, and so on)
- List of virtual devices (VDEVs) associated with this physical device (PDEV)
- Peer Id to Peer Object Mapping Array
- Receive processing information; for example, Rx re-order array
- Rx processing callback function registered by the OS interface layer
- TxRx host statistics
- Tx Descriptor Pool

#### **ol\_txrx\_vdev\_t: Object corresponding to TxRx virtual device**

This represents the transmit and receive characteristics in the offload mode of the virtual device with the corresponding OL physical device for a given VAP. A VDEV object corresponds to a network interface (such as *ath0*, *ath1*, and so on). Several VDEV objects can map to a single OL physical device (PDEV) as seen by the OS layer.

- Handle to the TxRx physical device (*ol\_txrx\_vdev\_t*)
- MAC address of the VAP
- Virtual device ID for this vdev
- List of peers associated with this VAP (vdev)
- Tx & Rx processing function pointers
- Rate Control Information (when done on host)
- Operational mode for the VAP

#### **ol\_tx\_desc\_t: Object corresponding to a Tx descriptor**

This represents a OL Tx descriptor. A pool of Tx descriptors are allocated during driver initialization. The size of pool is equal to 1K and is equal to total number transmit descriptors available on the target for a given radio interface. For every packet transmitted by the driver, a corresponding OL Tx descriptor is allocated. There is a unique ID associated with each OL Tx descriptor. This ID is passed as opaque meta-data to the target. The target returns this opaque ID of the Tx descriptor as part of Tx completion messages, which in turn is used by the host driver to free the Tx packet and the OL Tx descriptor.

**ol\_txrx\_peer\_t: Object corresponding to a peer node**

This structure stores important state information for an associated peer node. This data structure is mostly in the Rx path. Some of the important information stored in this structure are as follows:

- Handle to the VDEV object to which the peer is associated
- A list of peer IDs per peer. These peer IDs are used to obtain the corresponding node structure using the peer ID-to-node mapping
- MAC address of the VAP to which this peer is associated
- Rx re-order array per TID. This array is used to store Rx packets until the driver has received a continuous array of packets from the target, or a FLUSH message is received from the target. On receiving a FLUSH from the target, the whole array is sent up to the network stack, irrespective of whether the TID packet array has a continuum of packets.
- Security information

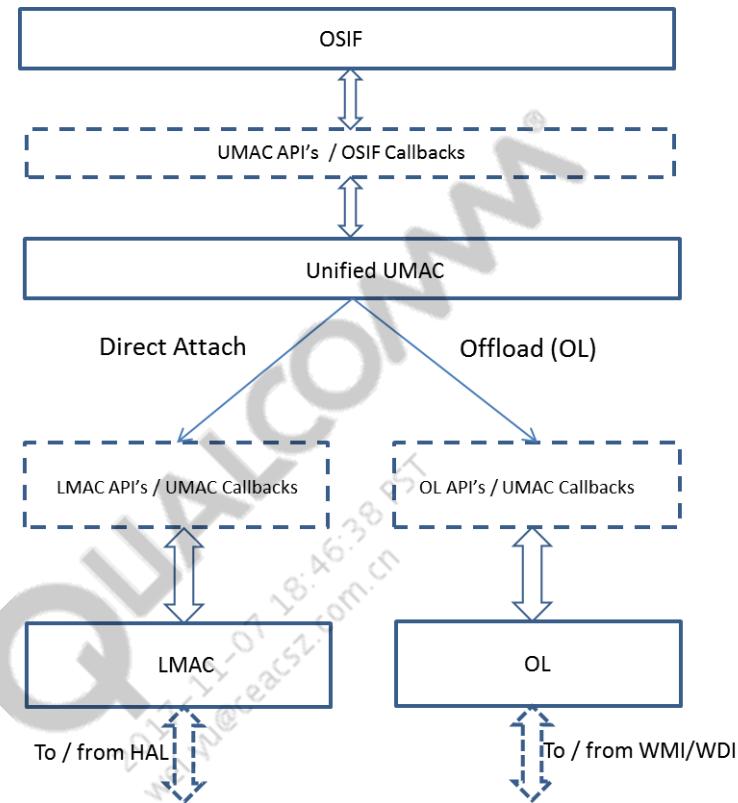
**cvg\_nbuf\_cb: Converged network buffer control block**

This is a very important meta-data structure stored either inline or as a pointer inside the OS specific data buffer structure (such as sk\_buff for linux, mbuf for NetBSD, NBLs for Windows). This meta-data structure maintains the following as part of every packet.

- scatter gather list sent down to the driver by the OS, as a fragment physical address & length.
- fragment physical address and length of the fragment added by the driver (HTT/HTC meta-data required by the target in the Tx path)

### 3.4.4 OL Data Structures

For offload based solutions, the OL (Off-Load) layer sits between the Unified MAC and the lower layers, such as the WMI (Wireless Management Interface) / WDI (Wireless Data Interface) interfaces to the target firmware. [Figure 3-17](#) captures this architecture:



**Figure 3-17 Offload Layer Positioning**

From [Figure 3-17](#) it is obvious that most of the OSIF layer & UMAC layer data structures are unchanged and can be re-used in case of the offload model. The LMAC and lower layers are never used in the offload model and are replaced with the OL and lower layers such as the WMI & WDI. The OL layer introduces new data structures, some of which are listed in the next section.

#### 3.4.4.1 ol\_ath\_softc\_net80211: OL radio device structure

The structure is typically referred to as “scn” in the code, and abstracts the radio interface device at the OL layer. In the offload architecture, there is a 1:1 mapping between the instance of the target firmware code and the radio interface. This data structure also captures and stores information about the instance of the target code running for this given radio interface. Like its counter-part in the direct-attach model, it embeds the UMAC layer ieee80211 structure. Some of the important fields in the structure as follows:

- UMAC common device structure (sc\_ic)
- OS
- device handle

- Callback registered by OS shim to be notified of target capabilities once the target is in service ready state.
- Some state information from target; for example, version/type/status
- Firmware download related information (BMI information)
- Handles to lower layers such as WMI, HTT, HTC, HIF
- Channel and Tx power information
- Structure to store TxRx statistics
- Locks for synchronization

### **3.4.4.2 ol\_ath\_vap\_net80211: OL network interface structure**

ol\_ath\_vap\_ieee80211 is the offload layer abstraction of the WLAN network interface. Each WLAN networks interface is represented as a “Virtual AP” (VAP) in the UMAC and shares a single underlying physical device/radio interface such as “*wifi0*”, “*wifi1*”, and so on. A VAP characterizes the operating mode of the access point which could be a host access point, an IBSS or a station. The important information stored in this structure are as follows:

- UMAC layer VAP object (av\_vap)
- The underlying OL layer physical device (OL layer scn)
- Handle to per VAP data-path structures (av\_txrx\_handle)
- Information used for beacons (such offload mode, beacon buffer, probe templates etc.)
- Timer, mostly required for cleanup.
- Lock for synchronization

### **3.4.4.3 ol\_ath\_node\_net80211: OL network interface structure**

ol\_ath\_node\_ieee80211 is the offload layer abstraction for an UMAC node. An UMAC node represents a connected station in the HOSTAP mode, an ad-hoc station in IBSS mode, and a BSS in infra-structure mode. The list of associated nodes represents the VAPs local view of the network. The OL node structure contains the following:

- UMAC layer node (an\_node)
- Handle to VAP data-path object (an\_txrx\_handle)

**NOTE** Other OL data structures such as in the modules like WMI / HTT / HTC / HIF and CE are assumed to have been covered in the respective sections.

## **3.5 Restore WLAN configuration after a firmware assert**

This section describes the functionality to perform a restoration of WLAN configuration when a firmware (FW) assert occurs. This functionality is introduced in the QCA\_Networking\_2017.SPF.5.0 release.

After a FW assert, a new capability is introduced to enable the WLAN subsystem to "not" reboot the AP, and, instead, collect the FW crash dump to restore the WLAN configuration. This restoration of WLAN settings occurs without removing the network interfaces of VAPs.

Configuration is partially restored by the WLAN driver itself, while the remainder is restored by the upper stack (hostapd, UCI, or customer applications). The recovery time is expected to be approximately five seconds. The implementation also needs to work in conjunction with any backend controller outside of the AP.

### 3.5.1 Radio and VAP restoration

The ol\_ath\_tasklet function detects the FW Assert and sets a flag in the iee80211com structure to indicate to the rest of the code that the target has crashed. The tasklet also disables further NAPI or tasklets from scheduling and does not continue to process the current tasklet. The tasklet schedules a worker thread to perform subsequent activities.

This worker thread performs the following tasks:

1. First, it allocates and creates a profile based on the current radio and all VAP configuration settings. All configuration parameters that are to be restored are saved in this profile data structure on a per-VAP basis. Apart from these parameters, a pointer to the VAP structure (ieee80211vap) is also stored for all VAPs.

**NOTE** The list of parameters to be restored is described in the "Configuration recovery list" section.

2. The worker thread iterates through all VAPs and brings down the network interface associated with them. The VAPs are also detached without unregistering the network interface and without freeing the VAP data structures, particularly the ieee80211vap structure. All data structures in the offload layer are freed including the "vdev" instance.
3. The worker thread calls the ol\_ath\_target\_stop API to detach the offload driver, without freeing the radio's network interface and the radio structures, especially the iee80211com structure. The offload driver detach includes freeing the "pdev" instance, destroying HTC, stopping wireless messaging interface (WMI), boot loader messaging interface (BMI) cleanup, and hardware interface (HIF) shutdown.
4. After the radio is stopped, the radio is restarted using the ol\_ath\_target\_start API. This API reattaches the offload driver and enables NAPI or tasklets, as needed.
5. After the radio is reattached, all the VAPs are re-created using their ieee80211vap structure and the network interface. Any configurations that are to be restored are used from the profile that was created in the first step. The profile structure is freed after being used.

### 3.5.2 DFS operations

During recovery, the dynamic frequency selection (DFS) state, which includes nonoccupancy list (NOL) and channel availability check (CAC) validity for current channel is preserved. After the channel availability check and channel selection are performed, the selected channel is stored in DFS structure and retained during Target assert recovery. If the user configures same channel

again after VAP is restored, CAC is omitted. The VAP bringup time is dependent on the application to push the configuration and bring the VAPs up. The time is also dependent on the number of VAPs.

### 3.5.3 Configuration recovery list

The following table lists the configuration parameters that are restored after a FW assert:

| Configuration parameter      | WLAN driver | Hostapd/UCI/Customer application |
|------------------------------|-------------|----------------------------------|
| VAP re-creation              | Yes         |                                  |
| VAP operation mode           | Yes         |                                  |
| VAP MAC address              | Yes         |                                  |
| DFS NOL list                 | Yes         |                                  |
| Security configuration       |             | Yes                              |
| All other WLAN configuration |             | Yes                              |

Apart from these configuration parameters, the WLAN driver also takes care of not waiting for CAC if the same channel is reissued by the upper layers.

### 3.5.4 Event notifications

The following two events are notified to the upper stack after a FW assert:

1. Target Assert Event—Sent as soon as the FW assert happens.
2. Target Reinit Event—Sent after all the VAPs are recovered. The upper stack is free to reapply the configuration.

## 3.6 Replace the direct calls to `printk` API with QDF print

Starting with QCA\_Networking\_2017.SP.F.5.0, the ASF based print framework on WIN is replaced with QDF print convergence. `printk` is replaced with QDF-based macros. The following design enhancements are made:

- The existing QDF framework implements generic QDF print function (such as `QDF_PRINT_INFO`)
- All instances of `printk` are replaced with this generic QDF print function
- QDF print function calls `printk` currently; with QDF Print convergences, it can be modified to call `QDF_TRACE`
- Automation scripts replace all `printk` to `QDF_PRINT_INFO`
- After all print replacements are done, `qdf_print` will be converted in the same way

## 3.7 Receive flow steering

This section describes the receive side scaling (RSS) and receive flow steering (RFS) capabilities that are supported on IPQ4019 platforms and other platforms that enable Ethernet subsystem (ESS) hardware flow steering.

RSS distributes packets by applying a hash function to each packet. Packets are steered to a separated receive ring based on the hash value. RSS steers packets solely based on hash, and thereby provides effective load distribution because it does not take into account application locality.

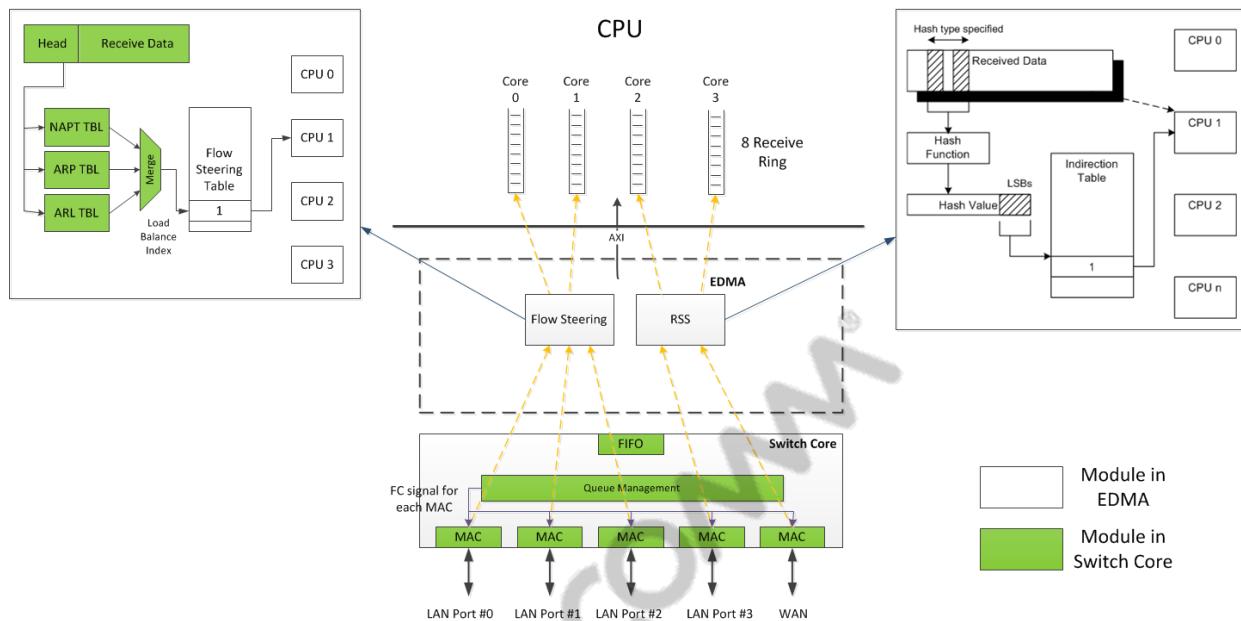
RFS is used to steer kernel processing of packets to the CPU where the application thread consuming the packet is running. RFS steers kernel processing of packets to the CPU where the application thread consuming the packet is running. That will increase data cache efficiency.

The switch for IPQ4019 supports both RSS and accelerated RFS. The following features are available:

- 4 Rx Rings and Interrupts (bind to 4 CPU cores).
- Flow Identification includes:
  - Destination MAC or VLAN + MAC (through ARL)
  - Destination IP (through ARP)
  - 5-tuple (through NAPT) – Flow Cookie
- Flow steered to different ring based on RSS/RFS result
  - RFS: Flow steered to different rings by configurations in each table entry.
  - RSS: Flow hashed to different rings by RSS hash value

The priority of IPQ4019 ESS RSS/RFS is as follows:

NAPT > ARP > ARL > RSS



**Figure 3-18 Flow steering in IPQ4019 ESS**

### 3.7.1 RPS and RFS in Linux kernel

Linux kernel has internal support for Receive Packet Steering (RPS) and accelerated RFS. RPS is logically a software implementation of RSS. This is accomplished by placing the packet on the desired CPU's backlog queue and waking up the CPU for processing. RPS is called during bottom half of the receive interrupt handler, when a driver sends a packet up the network stack with `netif_rx()` or `netif_receive_skb()`. These call the `get_rps_cpu()` function, which selects the queue that should process a packet.

While RPS steers packets solely based on hash, and thereby generally provides optimal load distribution, it does not take into account application locality.

This is accomplished by RFS. In RFS, packets are not forwarded directly by the value of their hash, but the hash is used as index into a flow lookup table. This table maps flows to the CPUs where those flows are being processed. If an entry does not hold a valid CPU, then packets mapped to that entry are steered using plain RPS.

For each hardware receive queue of the device, there is a `rps_dev_flow_table`. And there is global flow table `rps_sock_flow_table` that contains the \*desired\* CPU for flows: the CPU that is currently processing the flow in userspace. Each table value is a CPU index that is updated during calls to `recvmsg` and `sendmsg`.

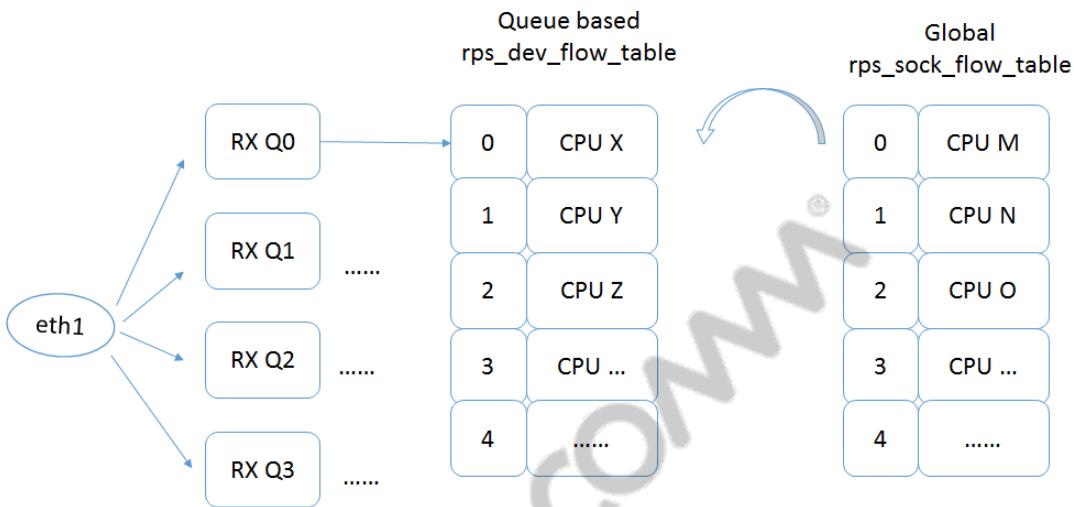


Figure 3-19 RPS/RFS in Linux kernel

### 3.7.2 Steering for packets from ESS to WLAN

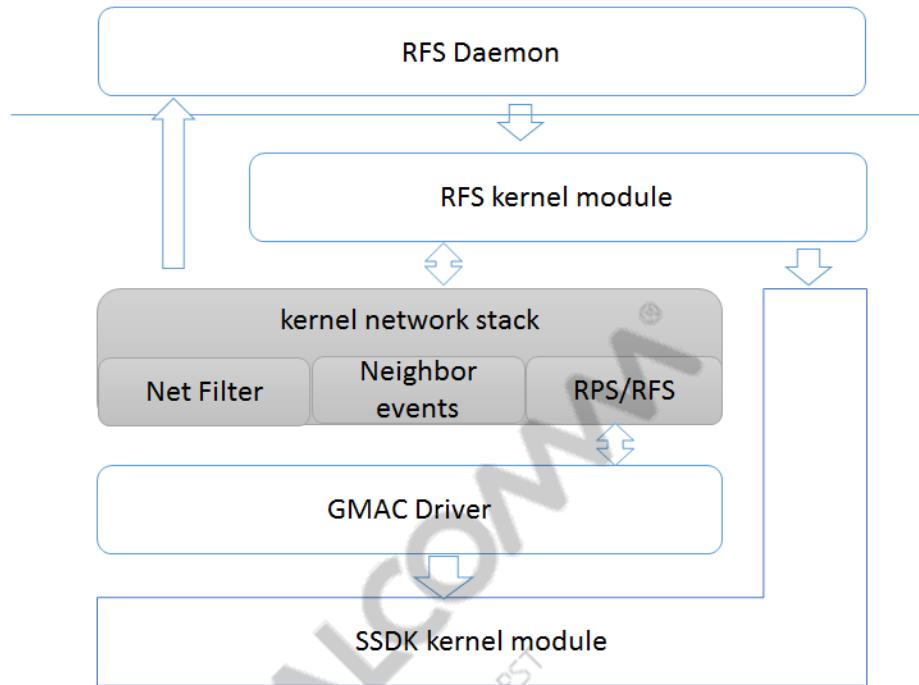
GMAC driver implements 5-tuple (or 4-tuple by kernel default) based RPS/RFS. The APIs between kernel and driver are defined using Linux kernel.

Packets to WLAN must be steered to specified CPU core according to WLAN band (2.4 GHz/5GHz).

Bridged packets are steered by its destination MAC address. Routed packets are steered by its destination IP address if the router is not configured with masquerade (SNAT). Otherwise, 5-tuple data is used to steer the incoming packets. The default steering method is RSS, even if kernel RPS/RFS is not enabled, hash-based RSS is used.

### 3.7.3 Interworking of components for RFS

The following diagram illustrates layered architecture of RFS and relationship between components:



- **GMAC driver**---Implements the kernel APIs which support RPS and accelerated RFS.
- **RFS kernel module**---A new kernel module, which identifies WLAN flows and setting RFS rules to kernel or SSDK module.
- **RFS daemon**---Running on user space for initializing RFS and receiving WLAN events from kernel.
- **SSDK Module**---Provides RSS/RFS APIs to program switch and EDMA.
- **Kernel Networking Stack**---Contains native RPS/RFS, neighbor events mechanism and netfilter support.

### 3.7.4 GMAC driver

GMAC driver implements Linux interface-based RPS/RFS. Linux kernel has defined the APIs between kernel and driver:

- An interface method: `ndo_rx_flow_steer()`

When this API is called, driver sets 5-tuple(or 4-tuple) based filter to SSDK, so that the flow is steering to specified CPU.

- A callback is called from driver: `rps_may_rexipre_flow()`

Driver periodically calls this function for each installed filter and remove expired filters.

Because kernel uses 4-tuple based filters by default. GMAC driver could overwrite the flow hash, that is, setting `skb->rxhash` with the hash of 5-tuple or other definition that could differentiate flows. GMAC driver maintains a local table for mapping SKB hash to hardware filters.

## 3.7.5 RFS module

RFS module is a new module, which recognizes WLAN flows, and set RFS rules to kernel or SSDK module. Therefore, it handles wireless events such as association/disassociation, process neighbor (IP address) changing events, trace IP flows, and set RFS rules. The IP/MAC based rules would be set through SSDK module, and the flow(5-tuple) based rules would be set though kernel rps\_sock\_flow\_table.

### 3.7.5.1 Database in RFS module

RFS module maintains a database of IP/MAC neighbors and receiving flows. At least three hash tables are present, namely, MAC table, IP (v4 and v6) table and IPv4 connection (5-tuple) table.

#### Steering MAC table

Each entry of MAC table has following structure:

```
struct rfs_mac_entry{
    MAC address;
    uint32 flag; /*indicate where the MAC address came from,
    2.4G/5G/user specified*/
    uint8   cpu;
}
```

The table entry is referenced by IP table entry.

#### IPv4/IPv6 address table

Each entry of MAC table has following structure:

```
struct rfs_ipv4/6_entry{
    IP address;
    unit32 flag; /*indicate if it a user specified IP entry*/
    uint8 cpu;
    struct rfs_mac_entry *mac; /*point to MAC address table entry*/
}
```

The table entry is referenced by IPv4 connection table entry.

#### IPv4 connection table

Because only DNATed flows (IPv4) coming from ESS must be processed, it is not necessary to maintain a full IPv4 connection table of system. The IPv4 connection entry is added according to the IP address. If a packet is DNATed to one of the address in IPv4 address table, the connection entry is created.

Each entry of connection table has following structure:

```
Struct rfs_ct_entry{
    src_ip, dest_ip, sport, dport;
    protocol;
    struct rfs_ipv4_entry *dest_ip_xlate; /*DNATED IPv4 address
    table entry, references to a IP table entry*/
    uint8 cpu;
}
```

## **WLAN STA join/leave events**

When WLAN STA is joining or leaving, WLAN driver sends netlink events to user space through kernel 802.11 stack. RFS daemon receives these events and pass them to RFS module.

The following are events handling of STA joining or leaving in RFS module:

### **Procedure of STA Joining**

1. Update or add MAC entry with specified CPU core according to the band. If the event comes from VLAN device, get the physical device from the real device instead of VLAN device.
2. Update or set MAC address rule to SSDK
3. If there are any IP address entries which referenced this MAC entry
 

```
{  
    update IP address entry with specified CPU core  
    update/set IP rules to SSDK  
    if there are any IP connection entries which referenced this IP entry, update rps_sock_flow_table so that Linux kernel could schedule the stream to the specified CPU  
}
```

### **Procedure STA Leaving**

1. Update MAC entry with CPU core “unspecified”. If the event comes from VLAN device, get the physical device from the real device instead of VLAN device.
2. Delete MAC rule from SSDK.
3. If any IP address entries that referenced this MAC entry exist:
 

```
{  
    update IP address entry with CPU core “unspecified”  
    delete the IP rule from SSDK  
    if there are any IP connection entries which referenced this IP entries, update rps_sock_flow_table so that Linux kernel could remove the flow from specified CPU  
}
```

### **3.7.5.2 IP neighbor change**

Linux kernel stack maintains IP neighbor information through ARP(IPv4) and ICMP6(IPv6). When a neighbor is reachable, stale or failed, kernel will sent messages through kernel event chains. When a IP neighbor is parsed successfully by ARP or ICMPv6, kernel will send “NUD\_REACHABLE” event. And when kernel failed to parse a neighbor, it sends the “NUD\_FAILED” event.

The following are the events that are handled when IP neighbor changes:

#### **New IP neighbor is added (reachable)**

- 1) Update or add IP entry
- 2) If this IP entry referenced to a MAC entry which has a specified CPU core, then set IP rule to SSDK with that CPU core
- 3) If there are any IP connection entries which referenced this IP entry, update the IP connection entry and rps\_sock\_flow\_table

#### **IP neighbor is deleted (failed)**

- 1) Update IP entry
- 2) If this IP entry referenced to a MAC entry which has a specified CPU core, remove the old IP rule from SSDK
- 3) If there are any IP connection entries which referenced this IP entry, update the IP connection entry and rps\_sock\_flow\_table

Deleting rule is delayed with an interval in cases in which the neighbor is added shortly thereafter.

#### **3.7.5.3 IP connection tracking**

This feature is only required for IPv4. Hooks are added on POST\_ROUTING to filter outgoing IPv4 packets. When a DNAT/SNAT flow is detected, the following operations occur:

1. Add IP connection entry.
2. If the DNATed IP address of this connection entry referenced to an IP entry which has a specified CPU core, then update rps\_sock\_flow\_table.

When Linux kernel removes a IP connection, it will send notification through kernel events chain. The following workflow occurs when one IP connection entry is removed:

1. Remove IP connection entry.
2. If this IP connection entry referenced an IP entry which has a specified CPU core, then clear the entry of rps\_sock\_flow\_table.

#### **3.7.5.4 RFS/RSS rules dependence and comparison**

In conclusion, IP connection rule depends on IP address rule, and IP address rule depends on MAC address rule. The following events occur if any rule is changed:

1. If a MAC rule is to be changed, change corresponding IP rules and IP connection rules.
2. If an IP rule is to be changed, change corresponding IP connection rules.
3. If an IP connection rule is to be changed, change the rule itself.

#### **3.7.5.5 Enable QRFS for VLAN Wireless Network Isolation**

When the wireless device is tagged with VLAN, when a new STA join or leave, it comes along with VLAN device in the event, so the real device is ascertained before the physical device is retrieved. Use system API is\_vlan\_dev to judge if it is a VLAN device. If it is a VLAN device,

then `vlan_dev_real_dev` is used to reach out to the real device, and real device is used to get the physical device and the target CPU number.

The comparison of RSS/RFS rules is as shown in following table:

| RSS/RFS Rules | Purposes                    | Rules based on             | Rules triggered by          | Rules set by                                                                                | Packets in-order delivery                                   | Rule Priority |
|---------------|-----------------------------|----------------------------|-----------------------------|---------------------------------------------------------------------------------------------|-------------------------------------------------------------|---------------|
| RSS Rules     | All IP packets              | Hash of 2-tuple or 4-tuple | Static                      | RFS module                                                                                  | Yes.<br>It's static.                                        | Extremely Low |
| MAC Rules     | Bridging packets            | Destination MAC address    | WLAN joining/Leaving Events | RFS module                                                                                  | Possible out-of-order when client has just joined           | Low           |
| IP Rules      | Routing packets without NAT | Destination IP address     | Kernel Neighbor Events      | RFS module                                                                                  | Possible out-of-order when a neighbor IP has just been got. | Medium        |
| 5-Tuple Rules | Routing packets with NAT    | 5-tuple of IP              | Post Routing Hook           | RFS updates Kernel <code>rps_sock_flow_table</code> , then rules will be set by GMAC driver | Yes.<br>Kernel ensures in-order delivery.                   | High          |

### 3.7.6 RFS daemon

RFS daemon is responsible for RFS initializing and WLAN event handling.

#### 1. RSS/RFS initialization

RFS daemon initializes the system with default RSS. The default RSS policy is to be determined.

#### 2. Obtain WLAN interrupt configuration

Get the CPU of WLAN interrupt processing, so that reversing flows are steered from ESS to WLAN to the same CPU as they were originally received.

#### 3. WLAN event handling

When a WLAN STA is joining or leaving, WLAN driver sends netlink events to user space through kernel 802.11 stack. RFS daemon will receive these events and pass them to RFS module.

Handling of events is performed in the following ways:

1. Receive STA joining/leaving event
2. Get band information of STA through 802.11 stack
3. Notify RFS module with joining or leaving of STAs

### 3.7.7 SSDK module APIs

SSDK provides the following APIs

#### RFS API for 5-tuple rule

```
int ssdk_rfs_ipct_rule_set(__be16 protocol, __be32 src_ip, __be32_
dest_ip, __be16 src_port, __be16 dest_port, uint8_t cpu);
int ssdk_rfs_ipct_rule_del(__be16 protocol, __be32 src_ip, __be32_
dest_ip, __be16 src_port, __be16 dest_port);
```

They will be called by GMAC driver.

#### RFS APIs for IPv4/IPv6 address rule

```
int ssdk_rfs_ip4_rule_set(__be32 dest_ip, uint8_t cpu);
int ssdk_rfs_ip4_rule_del(__be32 dest_ip);
int ssdk_rfs_ip6_rule_set(struct in6_addr *dest_ip, uint8_t cpu);
int ssdk_rfs_ip6_rule_del(struct in6_addr *dest_ip);
```

They will be called by RFS module.

#### RFS API for MAC address rule

```
int ssdk_rfs_mac_rule_set(uint8_t *dest_mac, uint8_t cpu);
int ssdk_rfs_mac_rule_del(uint8_t *dest_mac);
```

It will be called by RFS module.

#### RSS API for hash rule

```
#define RSS_HASH_TYPE_ENABLE 0x01
#define RSS_HASH_TYPE_IPV4_TCP_4TUPLE 0x02
#define RSS_HASH_TYPE_IPV6_TCP_4TUPLE 0x04
#define RSS_HASH_TYPE_IPV4_UDP_4TUPLE 0x08
#define RSS_HASH_TYPE_IPV6_UDP_4TUPLE 0x10
#define RSS_HASH_TYPE_IPV4_2TUPLE 0x20
#define RSS_HASH_TYPE_IPV6_2TUPLE 0x40
int ssdk_rss_rule_set(uint8_t flag );
```

Default UCI configuration and UCI initiating scripts are supported.

### 3.7.8 Sample configuration

The following is a sample configuration:

1. Enter the following commands to bring up ROOT AP.

```
uci set network.lan.ipaddr=192.168.1.3;
uci set network.Guest=interface
uci set network.Guest.type='bridge'
uci set network.Guest.proto='static'
uci set network.Guest.netmask='255.255.255.0'
uci set network.Guest.ipaddr='192.168.2.1'
uci commit network
uci set dhcp.Guest=dhcp
uci set dhcp.Guest.interface='Guest'
uci set dhcp.Guest.start='100'
```

```

uci set dhcp.Guest.limit='150'
uci set dhcp.Guest.leasetime='12h'
uci set dhcp.Guest.dhcpv6='server'
uci set dhcp.Guest.ra='server'
uci commit dhcp
/etc/init.d/network restart
/etc/init.d/dnsmasq restart

uci set repacd.repacd.Enable=1
uci set mcsd.config.Enable=0
uci set repacd.repacd.TrafficSeparationEnabled=1
uci set repacd.repacd.NetworkGuest=Guest
uci commit
/etc/init.d/repacd restart

```

2. Enter the following commands to bring up RE.

```

uci set network.lan.ipaddr=192.168.1.8;
uci set network.lan.gateway=192.168.1.3;
uci delete network.wan;
uci set network.Guest=interface
uci set network.Guest.type='bridge'
uci set network.Guest.proto='static'
uci set network.Guest.netmask='255.255.255.0'
uci set network.Guest.ipaddr='192.168.2.2'
uci commit network
uci set dhcp.Guest=dhcp
uci set dhcp.Guest.interface='Guest'
uci set dhcp.Guest.start='100'
uci set dhcp.Guest.limit='150'
uci set dhcp.Guest.leasetime='12h'
uci set dhcp.Guest.dhcpv6='server'
uci set dhcp.Guest.ra='server'
uci commit dhcp
/etc/init.d/network restart
/etc/init.d/dnsmasq restart

uci set repacd.repacd.TrafficSeparationEnabled=1
uci set repacd.repacd.NetworkGuest=Guest
uci set repacd.repacd.Enable=1
uci commit repacd
uci set wsplcd.config.DeepClone='0'
uci commit wsplcd
/etc/init.d/repacd start

```

3. Run the following commands on CAP and RE to make RE run the WPS handshake with CAP

```
env -i ACTION="pressed" BUTTON="wps" /sbin/hotplug-call button
```

4. Run traffic from the host behind the CAP and RE to check the RFS rule by using cat /proc/qrfb/rule, for the IP address from the VLAN Wi-Fi interface; the CPU must not be 65535.

## 3.8 Early detection of bit corruption during firmware assert

To enable an early detection of bit corruption to avoid firmware (FW) asserts, the internal RAM (IRAM) refresh mechanism is introduced. This functionality of early detection of bit corruption works in conjunction with the functionality to restore WLAN configuration during an FW assert.

The following are the advantages with the early detection of bit corruption:

- Early detection for IRAM
- A periodic CRC check by FW on IRAM section.
- Identify corruption early and reload before actual assert condition is triggered.
- Periodic FW IRAM refresh

The FW requests for the memory chunk to be allocated in the host DDR (crash dump location) to save a copy of IRAM in the WMI\_SERVICE\_READY\_EVENT. The host parses the payload of WMI\_SERVICE\_READY\_EVENT and responds with the WMI\_INIT\_CMD, which contains the host physical address signifying the location where the FW IRAM contents reside.

Upon the IRAM CRC failure or periodic IRAM download, FW restores the IRAM from host DDR using DMA, with the source address as host physical address. This source address is forwarded to FW from host using the WMI\_INIT\_CMD. A debug option is provided for the user to configure the timer to download the IRAM periodically, regardless of IRAM corruption.

During FW download of IRAM, based on the FW request of the memory chunk, the host writes a copy of IRAM. The FW uses this copy of IRAM when it detects a CRC failure on the IRAM.

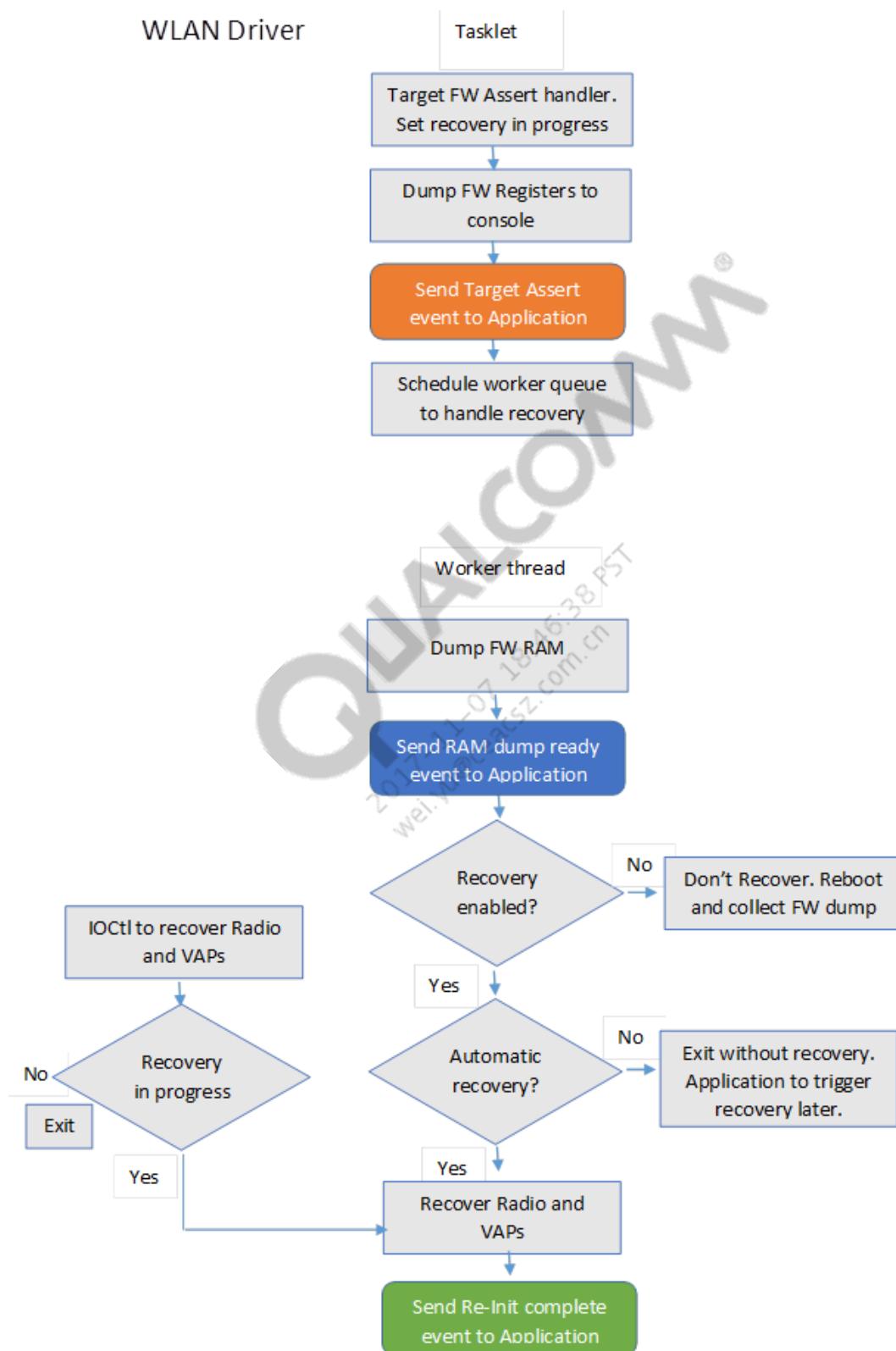
To avoid the IRAM write mechanism to overwrite the FW dump area, the FW recovery feature, after dumping the target RAM, waits for the user to continue the recovery process.

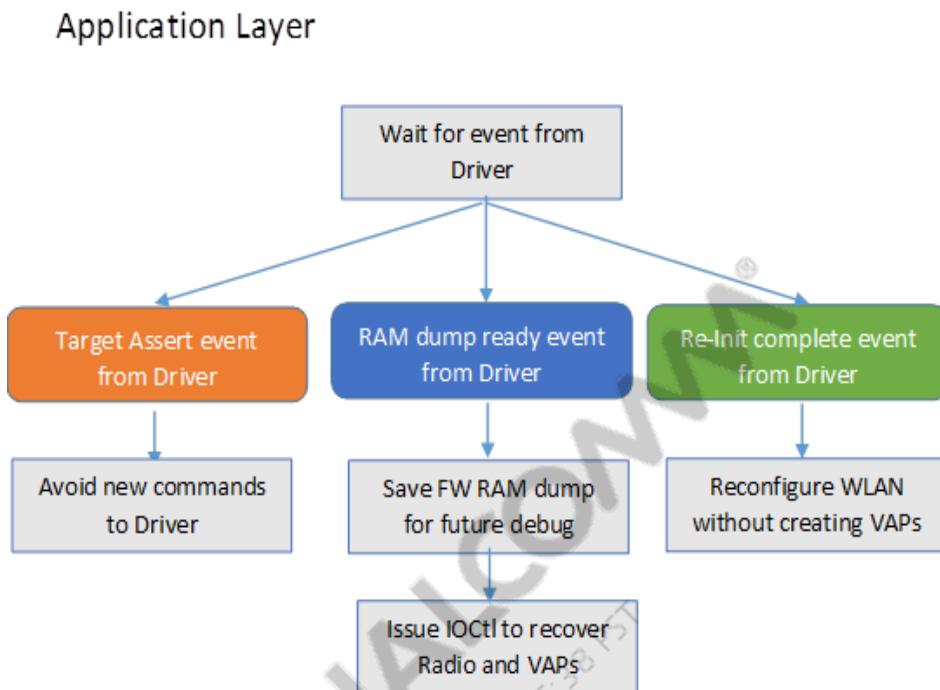
Enter the following command to enable the feature to detect bit corruption:

```
iwpriv wifiX set_fw_recovery <value>
```

- Value = 0, no FW recovery takes place. On target assert, after dumping data to RAM, the board is rebooted.
- Value = 1, (default) FW recovery restores all VAPs after RAM dump.
- Value = 2, FW recovery performs a RAM dump and waits for the user to trigger the complete recovery. If this ‘wait for user’ option is used, enter the ifconfig wifiX up command for the recover process to continue.

The following flowcharts illustrate this coexistence of FW recovery and IRAM write:





### 3.8.1 Target recovery parameters

To avoid the internal RAM (IRAM) write mechanism to overwrite the firmware (FW) dump area, the FW recovery feature, after dumping the target RAM, waits for the user to continue the recovery process.

| Parameter                            | Command                                                 | DA | OL | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------|---------------------------------------------------------|----|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>set_fw_recovery &lt;value&gt;</b> | <code>iwpriv wifiX set_fw_recovery &lt;value&gt;</code> | N  | Y  | <p>Enable the feature to detect bit corruption. The &lt;value&gt; option can be one of the following values:</p> <ul style="list-style-type: none"> <li>■ Value = 0, no FW recovery takes place. On target assert, after dumping data to RAM, the board is rebooted.</li> <li>■ Value = 1, (default) FW recovery restores all VAPs after RAM dump.</li> <li>■ Value = 2, FW recovery performs a RAM dump and waits for the user to trigger the complete recovery. If this 'wait for user' option is used, enter the following command for the recover process to continue:<br/><code>ifconfig wifiX up</code></li> </ul> |

# 4 WLAN AP Driver Build and Configuration Methods

---

This chapter describes the build options that are used with the makefile utility to construct and invoke the driver. It also discusses the runtime configuration methods, such as the wlanconfig utility and the configuration API (ACFG) that are required for proper device operations.

## 4.1 Build Options

### 4.1.1 Usage of build options

The following build options can be set when calling the makefile driver. A brief description of all supported command is provided in [Table 4-1](#).

**Table 4-1 Build Options**

| Options                      | Default | Description                                                                                                                                                                                                                                                                   |
|------------------------------|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ATH_BEACON_DEFERRED_PROC     | 0       | By default, beacons are prepared in an INTR context. This ensures the respect of timing requirements. Setting this option defers the beacon processing to the tasklet, which will decrease time in the interrupt, but might delay the beacons Tx under heavy load of the CPU. |
| ATH_BUS_PM                   | 0       | Enable the BUS Power Management feature. This will add the support of the <i>suspend</i> and <i>resume</i> sub-commands of <i>wlanconfig</i> into the driver.                                                                                                                 |
| ATH_CAP_AMSDU                | 1       | Enable the configuration of the Aggregated-MAC Service Data Unit (AMSDU) parameter.                                                                                                                                                                                           |
| ATH_CAP_TPC                  | 0       | Force the Transmit Power Control to be enabled on all chips. By default, it will be enabled on 11n chips, but disabled on 11g chips.                                                                                                                                          |
| ATH_DEBUG                    | 1       | Enable the debug support into the LMAC/HAL. When not defined, disables all the prints in the LMAC, regardless the run-time debug configuration.                                                                                                                               |
| ATH_GEN_TIMER                | 0       | Enables the use of the Generic Timer. Mainly used in P2P mode.                                                                                                                                                                                                                |
| ATH_IBSS_DFS_CHANNEL_SUPPORT | 0       | Enable this option to support DFS channel in AdHoc mode. If disabled, only non-DFS channels will be allowed in IBSS.                                                                                                                                                          |
| ATH_LMAC_TXSEQ               | 0       | Enable this option to fill the Tx sequence number in software prior sending the frame to the Hardware                                                                                                                                                                         |

**Table 4-1 Build Options (cont.)**

| Options                      | Default | Description                                                                                                                                                                                                                     |
|------------------------------|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ATH_MIB_INTR_FILTER          | 0       | This option can be enabled to filter-out the excessive MIB interrupts to avoid performance system issues.                                                                                                                       |
| ATH_NO_5G_SUPPORT            | 0       | If enabled, removes the 5G related code from the driver. Can be used to reduce code size on 2G only systems.                                                                                                                    |
| ATH_REGREAD_DEBUG            | 0       | Enables a message every time the driver tries to access one chip register                                                                                                                                                       |
| ATH_RFKILL                   | 0       | Enable this option to support the RFKILL feature through a GPIO. In that case, the driver will be regularly polling the GPIO pre-defined in the EEPROM and turn off the radio if required.                                      |
| ATH_RXBUF                    | 512     | Set the number of Rx buffers allocated by the system                                                                                                                                                                            |
| ATH_RXBUF_RECYCLE            | 0       | If set, recycles the Rx sk_buff directly in the driver without releasing them to the kernel first. This can have a significant benefit on CPU load.                                                                             |
| ATH_SHARED_IRQ               | 0       | Set this option on systems with multiple WLAN chipsets sharing the same IRQ.                                                                                                                                                    |
| ATH_SUPPORT_AGGR_BURST       | 1       | When enabled, this option increases the aggregation by adding a timer in Tx path in case other frames are coming. This can increase performance by increasing the aggregation rate, and therefore reducing the total bandwidth. |
| ATH_SUPPORT_AP_WDS_COMBO     | 0       | When enabled, the option "no_beacon" is available in the VAP iwpriv in order to disable the beacon transmission on this particular VAP.                                                                                         |
| ATH_SUPPORT_CFEND            | 0       | Enable this option to support and send CF-End frames as specified in 802.11.                                                                                                                                                    |
| ATH_SUPPORT_CWM              | 1       | When enabled, the driver integrates the <b>Channel Width Management</b> state machine to handle channel width selection dynamically.                                                                                            |
| ATH_SUPPORT_DESC_CACHABLE    | 1       | Allow the descriptors to be used and processed in cache memory. Disabling this option may have performance impact.                                                                                                              |
| ATH_SUPPORT_DESCFAST         | 1       | Process the descriptors in the ISR instead of differing them in tasklet context. Disabling this option may have a performance impact.                                                                                           |
| ATH_SUPPORT_DFS              | 1       | Support DFS algorithm. Disable this option if you want to use only non-DFS channels, or in single band system, to save space.                                                                                                   |
| ATH_SUPPORT_DYN_TX_CHAINMASK | 0       | When enabled, this feature forces the driver to limit the number of Tx chains to the number of spatial streams, for packets sent using a 64-QAM modulation.                                                                     |
| ATH_SUPPORT_EDMA             | 1       | Enable the Enhanced DMA feature.                                                                                                                                                                                                |
| ATH_SUPPORT_FLOWMAC_MODULE   | 0       | Enables the flow control support in the driver. When enabled concurrently with the flow control in QCA Ethernet driver, this can achieve some end-to-end flow control in the system.                                            |
| ATH_SUPPORT_GREEN_AP         | 1       | When enabled, adds some iwpriv to control the Green AP feature. This feature can disable some of the antennas when no station is connected in order to save power.                                                              |

**Table 4-1 Build Options (cont.)**

| Options                                | Default | Description                                                                                                                                                                                                                                |
|----------------------------------------|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ATH_SUPPORT_IBSS                       | 1       | Supports the AdHoc networking feature.                                                                                                                                                                                                     |
| ATH_SUPPORT_IBSS_ACS                   | 0       | Supports the ACS on AdHoc VAP.                                                                                                                                                                                                             |
| ATH_SUPPORT_IBSS_DFS                   | 0       | Supports DFS on AdHoc VAP.                                                                                                                                                                                                                 |
| ATH_SUPPORT_IBSS_HT                    | 0       | Allows usage of 11n on AdHoc VAP.                                                                                                                                                                                                          |
| ATH_SUPPORT_IBSS_NETLINK_NOTIFICATIONS | 0       | Enables the Netlink interface to monitor RSSI nodes status. User applications can subscribe to this API to monitor the other RSSI nodes status.                                                                                            |
| ATH_SUPPORT_IBSS_PERSTA_HWKEY          | 1       | When disabled, forces the usage of the same multicast key for all AdHoc nodes.                                                                                                                                                             |
| ATH_SUPPORT_IBSS_PRIVATE_SECURITY      | 0       | Enable this option to support security in AdHoc mode.                                                                                                                                                                                      |
| ATH_SUPPORT_IBSS_WMM                   | 0       | Supports WMM tagging on AdHoc VAP.                                                                                                                                                                                                         |
| ATH_SUPPORT_IQUE                       | 1       | Enables the IQUE features, to reduce latency and packet error rates for video transmissions.                                                                                                                                               |
| ATH_SUPPORT_IQUE_EXT                   | 0       | Disables some of the IQUE features (VoW and time-based retry) to decrease the CPU load.                                                                                                                                                    |
| ATH_SUPPORT_IWSPY                      | 0       | Enables the support of the wireless-tools "iwspy" command to monitor RSSI of connected stations.                                                                                                                                           |
| ATH_SUPPORT_LED                        | 0       | When enabled, this option activates the LED control within the driver. The WLAN chipset GPIO LEDs will be updated according to the traffic.                                                                                                |
| ATH_SUPPORT_LINUX_STA                  | 0       | Enables station mode support in the driver.                                                                                                                                                                                                |
| ATH_SUPPORT_LINUX_VENDOR               | 0       | When enabled, add two ioctls IDs to hook a standard ioctl call. This can be used to rewrite some private Get/Set functions that will be merged into the UMAC module.                                                                       |
| ATH_SUPPORT_PAPRD                      | 1       | Enable the dynamic PAPRD (Power Amplifier PRe-Distortion) calibration code for AR958x/AR959x-based designs. The PAPRD algorithm is used to reduce this signal distortion digitally using a combination of software and hardware processes. |
| ATH_SUPPORT_P2P                        | 0       | Enable WiFi peer-to-peer support in the driver.                                                                                                                                                                                            |
| ATH_SUPPORT_QUICK_KICKOUT              | 0       | When enabled, the AP kicks out a STA if several consecutive retries happen. The maximum consecutive retries can be customized using an iwpriv command.                                                                                     |
| ATH_SUPPORT_SPECTRAL                   | 0       | This option enables spectral scan feature support.                                                                                                                                                                                         |
| ATH_SUPPORT_TxBF                       | 0       | Enable this option to support transmit beamforming feature if available on chip.                                                                                                                                                           |
| ATH_SUPPORT_UAPSD                      | 1       | This option can be used to disable the UAPSD support in the driver.                                                                                                                                                                        |
| ATH_SUPPORT_UAPSD_RATE_CONTROL         | 0       | When enabled, this option will use a different rate control algorithm on a UAPSD-supported station to optimize the packet error rate on these stations.                                                                                    |
| ATH_SUPPORT_VLAN                       | 1       | Enable/disable VLAN support in the WLAN driver.                                                                                                                                                                                            |

**Table 4-1 Build Options (cont.)**

| Options                     | Default | Description                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------------------------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ATH_SUPPORT_VOWEXT          | 0       | This option enables the video over wireless (VoW) extensions in the driver. This must be used to enable the video improvements in the driver.                                                                                                                                                                                                                                                |
| ATH_SUPPORT_WAPI            | 0       | Enable this option to support WLAN authentication and privacy infrastructure (WAPI).                                                                                                                                                                                                                                                                                                         |
| ATH_SUPPORT_WPA2            | 1       | Supports WPA2 on an ad hoc VAP, as required by the 802.11i specification.                                                                                                                                                                                                                                                                                                                    |
| ATH_SW_WOW_SUPPORT          | 0       | Enables the wake on wireless (WoW) feature. When enabled, the driver builds the "WoW" kernel module.                                                                                                                                                                                                                                                                                         |
| ATH_SWRETRY                 | 0       | When enabled, this option schedules an additional software retransmission if all hardware retransmissions fail.                                                                                                                                                                                                                                                                              |
| ATH_TX_BUF_FLOW_CNTL        | 1       | This option enables video-focused Tx buffer allocation. Buffers for video/voice traffic are different than buffers for best effort (BE)/ background (BK) traffic. If no buffer are available from video traffic, then the driver will free buffers from BE/BK queues to avoid any impact on video.                                                                                           |
| ATH_TRAFFIC_FAST_RECOVER    | 1       | Enable the workaround for fast-recovery. It is observed that AP121 might stop responding sometimes due to PLL problems. When the device hangs, either transmission stops or transmission occurs at a wrong frequency. This condition requires a complete reset of the chip. This flag enables workaround for this condition. This makefile option is applicable only for AR934x-based chips. |
| ATH_VOW_EXT_STATS           | 1       | Enable certain iwpriv commands to access to VoW focus statistics.                                                                                                                                                                                                                                                                                                                            |
| ATH_WLAN_COMBINE            | 0       | Enable this option to reduce the number of modules by combining the WLAN modules together. When enabled, the UMAC, LMAC and Rate Control kernel modules will be built together into the LMAC module.                                                                                                                                                                                         |
| ATH_WPS_IE                  | 1       | Enables the usage of the WPS IE in the beacon. Disable this feature to remove the WPS IE from management frames at build time.                                                                                                                                                                                                                                                               |
| ATHEROS_LINUX_P2P_DRIVER    | 0       | Enables the direct connect (AKA P2P) feature in the driver.                                                                                                                                                                                                                                                                                                                                  |
| ATHEROS_LINUX_PERIODIC_SCAN | 1       | Enables some regular channels scanning in station modes. When disabled, a scan must be manually triggered to update the channel scanning information.                                                                                                                                                                                                                                        |
| BUILD_QDF                   | 1       | This option controls the build of the QDF layer. Disable this option to prevent the driver from building this module.                                                                                                                                                                                                                                                                        |
| BUILD_QDF_DEFER_PVT         | 1       | Disable this option to remove the DEFERRING part of the QDF layer.                                                                                                                                                                                                                                                                                                                           |
| BUILD_QDF_IRQ_PVT           | 1       | Disable this option to remove the IRQ part of the QDF layer.                                                                                                                                                                                                                                                                                                                                 |
| BUILD_QDF_NET               | 1       | Disable this option to remove the NET part of the QDF layer.                                                                                                                                                                                                                                                                                                                                 |
| CONFIG_RCPI                 | 0       | Supports the RCPI feature for T-DLS. Adds some iwpriv commands to control the RCPI parameters.                                                                                                                                                                                                                                                                                               |
| ENCAP_OFFLOAD               | 0       | Build USB offload support. Enable this feature to support USB offload chipsets.                                                                                                                                                                                                                                                                                                              |

**Table 4-1 Build Options (cont.)**

| Options                      | Default | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------------|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IEEE80211_MCAST_ENHANCEMENT  | 1       | Multicast traffic runs at very low bandwidth. Most of the time, multicast traffic is used for audio/video streaming. Some of the audio/video streams require higher bandwidth. Higher throughput can be achieved by using unicast frames for these applications. When enabled, all multicast traffic to wireless stations is either tunneled in (option 1) or translated ( option 2) to the unicast MAC address of the wireless station. The translation/tunnel happens with support of translation tables, which are built by learning through IGMP/V1/V2/V3 messages.<br><br><b>NOTE</b> No tunnel support for all partial offload chips. |
| NBUF_PREALLOC_POOL           | 0       | Use this option to pre-allocate WMI buffers for events and commands. This needs to be enabled for USB chipsets.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| NO_SIMPLE_CONFIG             | 0       | Enable this feature to disable the WSC button registration in the driver. If disabled, the WSC push button feature needs to be triggered through a third party application.                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| REMOVE_PKT_LOG               | 0       | Set this option to remove the packet log feature. This can be used to decrease the code size, but may decrease the ability to debug the driver.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| UMAC_SUPPORT_ACL             | 1       | Disable this option to remove the Access Control List feature from the driver. In that case, the driver cannot control which STA to authorize/reject at authentication time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| UMAC_SUPPORT_ACS             | 1       | Disable this option to remove Automatic Channel Selection algorithm in the driver.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| UMAC_SUPPORT_AP_POWERSAVE    | 1       | Disable this option to remove the Power Save support from the AP.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| UMAC_SUPPORT_APONLY          | 0       | Enable this option to remove all station-specific code from the UMAC and decrease its size.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| UMAC_SUPPORT_CCMP_SW_CRYPTO  | 1       | Disable this option to remove the CCMP encryption software from the driver. In that case, only hardware encryption (into the WLAN chip) will be available.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| UMAC_SUPPORT_DFS             | 1       | Disable this option to remove DFS support from the UMAC.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| UMAC_SUPPORT_IBSS            | 1       | Disable this option to remove ad hoc support from the UMAC.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| UMAC_SUPPORT_IE_UTILS        | 0       | Enable this option to have build code to access IEs easily using a dedicated parsing function. This is currently mainly used by P2P code.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| UMAC_SUPPORT_NAWDS           | 1       | Disable this option to remove the Non-Associated WDS feature from the driver.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| UMAC_SUPPORT_POWERSAVE_QUEUE | 1       | When set, this option adds the support of a dedicated queue for Power Save management.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| UMAC_SUPPORT_RESMGR          | 0       | Enable this option to build the P2P resource management                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| UMAC_SUPPORT_RESMGR_OC_SCHED | 0       | Enable this option to build the P2P resource management scheduling.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| UMAC_SUPPORT_RESMGR_SM       | 0       | Enable this option to build the P2P resource management state machine.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

**Table 4-1 Build Options (cont.)**

| Options                                    | Default | Description                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------------------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| UMAC_SUPPORT_RIJNDAEL                      | 1       | Disabling this option removes the RIJNDAEL cypher code from the driver.                                                                                                                                                                                                                                                                                                                                                                 |
| UMAC_SUPPORT_RPTPLACEMENT                  | 0       | Set this option to add the Repeater Placement feature in the UMAC.                                                                                                                                                                                                                                                                                                                                                                      |
| UMAC_SUPPORT_SMARTANTENNA                  | 0       | Enable this option to add Smart Antenna support in the UMAC.                                                                                                                                                                                                                                                                                                                                                                            |
| UMAC_SUPPORT_STA_POWERSAVE                 | 1       | Disable this option to remove the powersave feature in station mode.                                                                                                                                                                                                                                                                                                                                                                    |
| UMAC_SUPPORT_STAFWD                        | 0       | Enable this option to add the Station Forwarding feature into the UMAC. This option allows the connection of a PC behind the station.                                                                                                                                                                                                                                                                                                   |
| UMAC_SUPPORT_TKIP_SW_CRYPTO                | 1       | Disable this option to remove the TKIP software compression algorithm from the driver. In that case, only hardware encryption will be available.                                                                                                                                                                                                                                                                                        |
| UMAC_SUPPORT_TSF_TIMER                     | 0       | Enable this option to allow the VAP to schedule timer events based on the hardware TSF clock. This is mainly used in P2P feature.                                                                                                                                                                                                                                                                                                       |
| UMAC_SUPPORT_TX_FRAG                       | 1       | Disable this option to drop Tx frames bigger than the fragmentation threshold. If option is enabled (default), the packet will be fragmented.                                                                                                                                                                                                                                                                                           |
| UMAC_SUPPORT_VAP_PAUSE                     | 0       | Enable this option to support VAP pause in P2P mode. This feature is mainly used in P2P mode when multiple VAPs are used.                                                                                                                                                                                                                                                                                                               |
| UMAC_SUPPORT_VI_DBG                        | 0       | Adds video focused debug features into the UMAC.                                                                                                                                                                                                                                                                                                                                                                                        |
| UMAC_SUPPORT_WDS                           | 1       | Disable this option to remove WDS support from the UMAC.                                                                                                                                                                                                                                                                                                                                                                                |
| VOW_LOGLATENCY                             | 0       | Update the driver TSF structure on a per-aggregate basis rather than on a per-packet basis. This can be useful and could reduce the CPU load in case precise VoW logging capability is not needed.                                                                                                                                                                                                                                      |
| VOW_TIDSCHED                               | 0       | Enable the VoW specific queue scheduling. When this option is enabled, the queue weight can then be configured using dedicated iwpriv commands.                                                                                                                                                                                                                                                                                         |
| QCA_OL_SUPPORT_RAWMODE_TXRX                | 0       | Enable/disable raw mode Tx/Rx <ul style="list-style-type: none"> <li>■ 0—Disable</li> <li>■ 1—Enable</li> </ul>                                                                                                                                                                                                                                                                                                                         |
| QCA_OL_SUPPORT_RAWMODE_AR900B_BUFFCOALESCE | 0       | Enable/disable raw mode Tx buffer coalescing for QCA9980 if number of buffers exceeds 6. <p style="text-align: center;"><b>NOTE</b> This should be disabled to save CPU cycles on end platforms which can take care of ensuring that number of buffers doesn't cross 6 before the skbs reach the WLAN driver (say using jumbo frames or NSS copy)</p> <ul style="list-style-type: none"> <li>■ 0—Disable</li> <li>■ 1—Enable</li> </ul> |

**Table 4-1 Build Options (cont.)**

| Options                            | Default | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------------------------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| QCA_RAWMODE_OPTIMIZATION_CONFIG    | 0       | <p>Optimization configuration for raw mode. This configures inlining and compiler likeliness hints for 'if' conditions. Doesn't add/remove functionality.</p> <p><b>NOTE</b> This should be set to 1 or 2 according to the desired use cases, on end platforms.</p> <ul style="list-style-type: none"> <li>■ 0—Assume that raw mode will mostly not be run time enabled in production</li> <li>■ 1—Assume that only raw mode will mostly be run time enabled in production</li> <li>■ 2—Assume that mixed mode (some VAPs raw, some VAPs non-raw) can be configured at run time in production</li> </ul> |
| QCA_SUPPORT_RAWMODE_PKT_SIMULATION | 0       | <p>Enable/disable raw mode simulation at driver entry/exit points.</p> <p><b>NOTE</b> This should be disabled on end systems</p> <ul style="list-style-type: none"> <li>■ 0—Disable</li> <li>■ 1—Enable</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                       |
| PEER_CACHEING_HOST_ENABLE          | 1       | <p>Enable/disable QCache feature.</p> <ul style="list-style-type: none"> <li>■ 0—Disable</li> <li>■ 1—Enable</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

## 4.2 Runtime configuration options

The QCA WLAN driver is divided into different kernel modules. After inserting all these modules, the WLAN driver should automatically detect the QCA WLAN chipsets, and create one Linux network interfaces per chipset detected.

The applications below use device names to determine which device to configure. In the QCA WLAN driver, two device types are created when the AP is brought up. The radio, or LMAC layer is instantiated as a **wifiN** device, where N is the specific instance starting with zero. The first radio instance, for example, would be called wifi0.

The protocol, or UMAC layer, is instantiated as an **athN** device. These devices are also known as virtual APs (VAPs). Multiple VAPs can be associated with a single radio, but only one radio associated with any particular VAP (one-to-many relationship), as described in the top-level architecture. Each layer controls specific aspects of system operation, so specific commands apply to each.

This chapter provides a brief overview of the run-time configuration that is required for proper device operation. For more information on these operations, please refer to the *Wireless LAN Access Point, Driver Version 10.4, Command Reference Guide* (80-Y8052-1).

### 4.2.1 wlanconfig Utility

The QCA wlanconfig utility manages VAP instances. It is an integral part of the configuration process and provides the primary method to:

- Create VAPs
- List VAP parameters
- Delete interfaces

#### 4.2.1.1 Create a VAP interface

Creating a VAP requires parameters indicating the specific nature of the VAP. A VAP can be either a client node (managed or Station), an infrastructure node (master or Access Point), or an IBSS node (AdHoc). A monitor VAP can also be created to listen to the WLAN traffic on this particular radio.

The commands below can be used to create some VAP instance:

**Table 4-2 VAP commands**

| Mode         | Format                                                                              | Description                                                                 |
|--------------|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|
| AP mode      | wlanconfig <b>athN</b> create wlandev <b>wifiN</b><br>wlanmode <b>ap</b>            | Create VAP <b>athN</b> on top of <b>wifiN</b> interface in <b>AP</b> mode.  |
| Station mode | wlanconfig <b>athN</b> create wlandev <b>wifiN</b><br>wlanmode <b>sta</b> nosbeacon | Create VAP <b>athN</b> on top of <b>wifiN</b> interface in <b>STA</b> mode. |

**Table 4-2 VAP commands (cont.)**

|              |                                                                             |                                                                                 |
|--------------|-----------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| Ad Hoc mode  | wlanconfig <b>athN</b> create wlandev <b>wifiN</b><br>wlanmode <b>adhoc</b> | Create VAP <b>athN</b> on top of <b>wifiN</b> interface in <b>Ad Hoc</b> mode.  |
| Monitor mode | wlanconfig <b>monN</b> create wlandev<br><b>wifiN</b> wlanmode <b>mon</b>   | Create VAP <b>athN</b> on top of <b>wifiN</b> interface in <b>Monitor</b> mode. |

The following parameters can be customized depending on the expected configuration:

- **athN**: name of the VAP interface to create. If the command succeeds, a new Linux network device interface with that name will be created. This will be the interface used to connect with IP network stack, and used to configure the VAP parameters.
- **wifiN**: indicates the interface to which the VAP will attach. The interface number is required for the argument. This is a mandatory argument, as in systems where multiple radio are connected, this will represent the radio to use for this particular VAP.
- **[ap|sta|adhoc|mon]**: indicates the mode for the VAP. *ap* will create an AP VAP, *sta* will create an STA, *adhoc* will create an AdHoc VAP and *mon* will create a monitor VAP.

#### 4.2.1.2 List VAP parameters

The list command can be used to get information about the VAP status. It can also be used to obtain information about devices connected to this VAP such as rate, capabilities, beacon interval, SSID of the detected APs (for STA VAP); or the rate used for this STA, current Tx/Rx Sequence number, capabilities, and current state (for STAs).

**Table 4-3 List VAP Command**

| Command                     | Description                                                                |
|-----------------------------|----------------------------------------------------------------------------|
| wlanconfig <b>athN</b> list | List all available information about devices connected to <b>athN</b> VAP. |

#### 4.2.1.3 Delete a VAP interface

A VAP interface can be deleted by using the following command. The Linux network device must be down prior running this command.

**Table 4-4 Delete VAP Command**

| Command                        | Description                  |
|--------------------------------|------------------------------|
| wlanconfig <b>athN</b> destroy | Delete the VAP <b>athN</b> . |

#### 4.2.2 Wireless Tools

The wireless tools interface is used to configure and operate the WLAN interface. The tools themselves are open source, and require specific supports in the driver through the ioctl interface. The QCA WLAN driver supports these tools and they are used as the main configuration interface.

The two main programs of the wireless tools suite are **iwconfig** and **iwpriv**. These commands are used to get or change specific configuration or system operating parameters. Many commands are

only effective before the AP interface is in the up state, so these commands must be performed prior to issuing the **ifconfig up** command to the interface. This document defines the valid parameters and commands for each command. Note that the radio layer does not support the **iwconfig** command, which is used exclusively in the protocol layer. Also, note that any **ifconfig** command used on the AP must be applied to the protocol layer (ath) device. Any packet directly sent from the IP stack to the radio layer will be dropped by the driver as it will not contain any protocol information.

#### 4.2.2.1 iwconfig

The **iwconfig** command defines a fixed set of parameters used to set up and operate the major WLAN features. They are used in much the same way as **ifconfig** commands but are specific to 802.11 device operations. Thus, they interface to the particular VAP interface. Note that the Radio Layer does not support **iwconfig**.

Below is a brief description of the commands you can find in **iwconfig**. More details can be found in the *AP CLI User Guide* documentation, or in the **iwconfig** manual.

**Table 4-5 Iwconfig Commands**

| Parameter    | Format                                                                   | Description                                                                                                                                                 |
|--------------|--------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ap           | <b>iwconfig athN ap MAC</b>                                              | Connect VAP <b>athN</b> to the AP with BSSID <b>MAC</b>                                                                                                     |
| channel freq | <b>iwconfig athN channel CHANNEL</b><br><b>iwconfig athN freq FREQ</b>   | Set the radio below VAP <b>athN</b> on channel <b>CHANNEL</b> (if channel command is used) or on frequency <b>FREQ</b> (if freq command is used).           |
| enc key      | <b>iwconfig athN enc key_value</b><br><b>iwconfig athN key key_value</b> | Set the WEP key to <b>key_value</b> for the VAP <b>athN</b>                                                                                                 |
| essid        | <b>iwconfig athN essid NAME</b>                                          | Set the essid for VAP <b>athN</b> to <b>NAME</b>                                                                                                            |
| frag         | <b>iwconfig athN frag level</b>                                          | Set the fragmentation threshold to <b>level</b> on VAP <b>athN</b>                                                                                          |
| rate         | <b>iwconfig athN rate RATE</b><br><b>iwconfig athN rate auto</b>         | Force the rate to <b>RATE</b> for non-HT transmission, on VAP <b>athN</b> . <b>auto</b> can be used as a rate argument to reset the VAP rate configuration. |
| rts          | <b>iwconfig athN rts pkt_size</b>                                        | Sets the minimum packet size for which RTS/CTS protection is used. The VAP <b>athN</b> will then use <b>pkt_size</b> as a maximum size.                     |
| txpower      | <b>iwconfig athN txpower PWR</b>                                         | Sets the maximum TxPower to use on the radio under <b>athN</b> to <b>PWR</b> (unit is expressed in dBm).                                                    |

#### 4.2.2.2 iwpriv

The **iwpriv** commands are used to configure almost all 11n features and QCA proprietary features in the WLAN drivers, organized per layer. All VAP configuration related parameters can be configured using **iwpriv** on the VAP interface, and all Radio configuration related parameters can be configured using **iwpriv** on the Radio interface.

Commands are available to configuration the following (this list is non-exhaustive):

- Aggregation
- Association/ACL

- Beacons
- Channel width
- Debug
- Physical layer
- Protection mechanism
- VoW
- WMM

Please refer to the *AP CLI User Guide*, MKG-15363 for the full commands list.

## 4.3 Configuration API (ACFG)

The configuration API provides a set of library functions available in user space through which a user application can configure the WLAN driver and hostap (hostapd and supplicant). In addition, a user application may also register for events from the WLAN driver and hostap application. The main objective of the configuration API is provide a uniform configuration interface to the user application and hide all the configuration dependencies. The configuration API functions finds all the dependencies and appropriately configures the WLAN driver and hostap in the correct sequence. These APIs could be integrated in to a customer application that configures a complete system.

The following are the configurations that are currently supported through ACFG:

- Basic AP and STATION profile configuration
- Security modes: Open, WEP, WPA/WPA2 personnel
- Multiple VAPs
- AP + STA configuration
- Dual Radio Profile configurations
- WPS
- 802.1x
- WDS
- VLAN
- Event Log

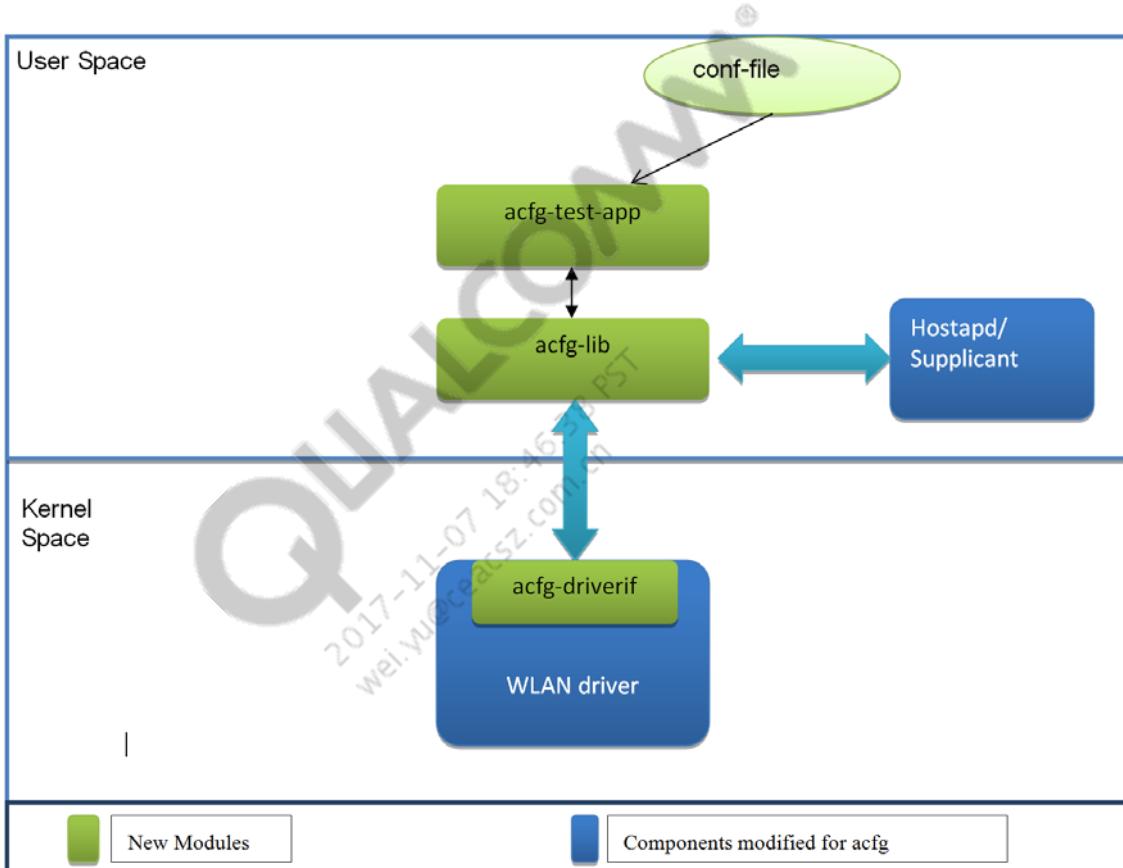
### 4.3.1 Theory of Operation

#### 4.3.1.1 Overall Architecture

The Configuration API has the following software components:

- acfg-lib: A configuration API library module in user space that provides the APIs for configurations and events.
- acfg-test-app: A sample acfg test application that takes the configuration profile stored in a file as input and appropriately calls the APIs provided by the acfg-lib.
- acfg-driverif: This module implements the ACFG ioctl command handler and ACFG event handler, and interfaces with the WLAN driver ioctl and OS interface modules.

Figure 3-1 shows the overall architecture of configuration API support.



**Figure 4-1 Configuration API Support Architecture**

### Acfg-lib

The APIs provided by acfg-lib module are classified in to two main categories: Configuration APIs and Events API.

- **Configuration APIs**

The configuration APIs provide profile-based configuration methods for the user application. Configuration for each WLAN device is represented as a profile and provided as input to these APIs. The WLAN device configurations gets created, deleted, modified or retrieved through these APIs.

- **Events API**

This API allows user applications to receive events from the WLAN driver and hostap application. Events from the WLAN driver and hostap application are sent to the Netlink socket and the API listens on the netlink socket and calls appropriate callback functions registered by the user application when these events are received.

### **Acfg-testapp**

This module is a sample test application provided to test the Config API functionalities. This test app takes the profile configuration file as an input and appropriately calls the config API provided by the acfg-lib module.

### **Acfg-driver interface**

The Acfg-driver interface is a Linux OS interface that handles the ACFG private ioctl for both radio and network interfaces. All the ACFG commands are sent to the driver through ACFG\_PVT\_IOCTL. The interface also handles the ACFG events from the driver and sends it to ACFG application.

## **4.3.2 Implementation**

### **4.3.2.1 ACFG APIs Library**

This module is implemented under the apps/acfg/src and apps/acfg/include directories. The following are the major data structures and major API implementation details.

#### **Major Data Structure**

- **acfg\_wlan\_profile\_t**

Acfg wlan profile structure change to accomodate all VAPs per radio

```
typedef struct {
    char ctrl_hapd[ACFG_CTRL_IFACE_LEN];
    char ctrl_wpasupp[ACFG_CTRL_IFACE_LEN];
    acfg_wlan_profile_radio_params_t radio_params;
    acfg_wlan_profile_vap_params_t vap_params[ACFG_MAX_VAPS];
    a_uint8_t num_vaps;
} acfg_wlan_profile_t;
```

- **acfg\_wlan\_profile\_radio\_params\_t**

```
typedef struct {
    a_uint8_t radio_name[ACFG_MAX_IFNAME]; /* key */
    a_uint8_t chan; /* read-write, mandatory */
    a_uint32_t freq; /* in Mhz read-write, mandatory */
    a_uint16_t country_code; /*read-write, optional; 0 if unspecified */
    a_uint8_t radio_mac[ACFG_MACSTR_LEN];
        /* read-write, optional; 0 if unspecified */
    a_uint8_t ampdu[32];
    a_uint32_t ampdu_limit_bytes;
    a_uint8_t ampdu_subframes;
} acfg_wlan_profile_radio_params_t;
```

### ■ acfg\_wlan\_profile\_vap\_params\_t

```
typedef struct {
    a_uint8_t vap_name[ACFG_MAX_IFNAME]; /* key */
    a_uint8_t radio_name[ACFG_MAX_IFNAME]; /* key */
    acfg_opmode_t opmode; /* read-write, mandatory */
    acfg_phymode_t phymode; /* read-write, mandatory */
    a_uint8_t ampdu[32];
    a_uint8_t ssid[ACFG_MAX_SSID_LEN]; /* read-write, mandatory */
    a_uint32_t bitrate; /* read-write, mandatory */
    a_uint8_t rate[16]; /* read-write, mandatory */
    a_uint32_t retries;
    acfg_txpow_t txpow; /* read-write mandatory */
    acfg_rssi_t rssi; /* read-only */
    a_uint32_t beacon_interval; /* read-write, optional; 0 if unspecified */
    acfg_rts_t rts_thresh; /* read-write, optional; 0 if unspecified */
    acfg_frag_t frag_thresh; /* read-write, optional; 0 if unspecified */
    a_uint8_t vap_mac[ACFG_MACSTR_LEN];
    acfg_wlan_profile_security_params_t security_params;
    acfg_wlan_profile_node_params_t node_params; acfg_wds_params_t wds_params;
    a_uint32_t vlandid;
    a_uint8_t bridge[ACFG_MAX_IFNAME];
        /* read-write, optional; 0 if unspecified */
    a_uint32_t pureg;
    a_uint32_t puren;
    a_uint32_t hide_ssid;
    a_uint32_t doth;
    a_uint32_t client_isolation; a_uint8_t coext[32];
    a_uint32_t uapsd;
    a_uint8_t shortgi[32];
} acfg_wlan_profile_vap_params_t;
```

### ■ acfg\_wlan\_profile\_security\_params\_t

```
typedef struct {
    acfg_wlan_profile_sec_meth_e sec_method; /* read-write, mandatory */
    acfg_wlan_profile_cipher_meth_e cipher_method; /* read-write, mandatory */
    acfg_wlan_profile_cipher_meth_e g_cipher_method; /* read-write, mandatory */
    a_uint8_t psk[ACFG_MAX_PSK_LEN];
        /* read-write, mandatory if sec_method = WPA/WPA2/WPS, 0 otherwise */
    a_uint8_t wep_key0[ACFG_MAX_WEP_KEY_LEN];
        /* read-write, mandatory if sec_method = WEP, 0 otherwise */
    a_uint8_t wep_key1[ACFG_MAX_WEP_KEY_LEN];
        /* read-write, optional, 0 if unspecified */
    a_uint8_t wep_key2[ACFG_MAX_WEP_KEY_LEN];
        /* read-write, optional, 0 if unspecified */
    a_uint8_t wep_key3[ACFG_MAX_WEP_KEY_LEN];
        /* read-write, optional, 0 if unspecified */
    a_uint8_t wep_key_defidx;
    a_uint32_t wps_pin; /* write-only, 0 if unspecified */
    a_uint8_t wps_flag;
    a_uint32_t wpa_group_rekey;
    a_uint8_t wps_upnp_iface[ACFG_MAX_IFNAME];
    a_uint8_t wps_friendly_name[ACFG_WSUPP_PARAM_LEN];
    a_uint8_t wps_man_url[ACFG_WSUPP_PARAM_LEN];
    a_uint8_t wps_model_desc[ACFG_WSUPP_PARAM_LEN];
```

```

    a_uint8_t    wps_upc[ACFG_WSUPP_PARAM_LEN];
    a_uint32_t   wps_pbc_in_m1;
    a_uint8_t    wps_device_name[ACFG_WSUPP_PARAM_LEN];
    acfg_wlan_profile_sec_eap_params_t eap_param;
    acfg_wlan_profile_sec_radius_params_t radius_param;
} acfg_wlan_profile_security_params_t;

typedef enum {
    ACFG_EAP_TYPE_PEAP = 1,
    ACFG_EAP_TYPE_LEAP,
    ACFG_EAP_TYPE_TLS,
} EAP_TYPE;

typedef struct {
    a_uint8_t    eap_type;
    a_uint8_t    identity[128];
    a_uint8_t    password[128];
    a_uint8_t    ca_cert[512];
    a_uint8_t    client_cert[512];
    a_uint8_t    private_key[512];
    a_uint8_t    private_key_passwd[128];
} acfg_wlan_profile_sec_eap_params_t;

typedef struct {
    a_uint8_t    radius_ip[IP_ADDR_LEN];
    a_uint32_t   radius_port;
    a_uint8_t    shared_secret[RADIUS_SHARED_SECRET_LEN];
} acfg_wlan_profile_sec_radius_params_t;

■ acfg_wlan_profile_node_params_t

typedef struct {
/*
 * This list is to add mac addresses to the first ACL list
 */
    a_uint8_t    acfg_acl_node_list[ACFG_MAX_ACL_NODE][ACFG_MACADDR_LEN];
    a_uint8_t    num_node;
    acfg_wlan_profile_node_acl_t node_acl; /* Set acl policy of first ACL
list*/
/*
 * This list is to add mac addresses to the secondary ACL list
 */
    a_uint8_t    acfg_acl_node_list_sec[ACFG_MAX_ACL_NODE][ACFG_MACADDR_
LEN];
    a_uint8_t    num_node_sec;
    acfg_wlan_profile_node_acl_t node_acl_sec; /* Set acl policy of
secondary ACL list*/
}

} acfg_wlan_profile_node_params_t;

■ acfg_wds_params_t

typedef struct {
    a_uint8_t    enabled;
    a_uint8_t    wds_addr[ACFG_MACADDR_LEN];
    a_uint32_t   wds_flags;
} acfg_wds_params_t;

```

```

typedef enum {
    ACFG_FLAG_VAP_IND = 1,
    ACFG_FLAG_EXTAP = 2,
} WDS_FLAGS;

■ wps
typedef struct {
    a_uint8_t wps_state;
} acfg_hapd_param_t;

enum acfg_event_handler_type {
    ACFG_EVENT_WPS_NEW_AP_SETTINGS = 1,
    ACFG_EVENT_WPS_SUCCESS,
};

■ VLAN

```

```

typedef struct acfg_vlangrp_info {
    acfg_dl_cfg_hdr_t dl_cfg_hdr;
    char if_name[ACFG_MAX_IFNAME];
    char vlan_id[ACFG_MAX_VIDNAME];
} acfg_vlangrp_info_t;

```

## Major APIs

### ■ **acfg\_set\_profile**

- Description

This API is used to

- Create VAPs
- Delete VAPs
- Modify Parameters
- Radio Parameters
- Modify Node ACLs

The decision will be taken based on the difference in the current driver parameters and the new parameters

- Get current parameters from the driver
  - List of created VAPs
  - List of ACL and policies per VAP
- Compare the VAP names as given in the arguments and driver, and determine the action to perform
  - Create a new VAP in case the supplied VAP name does not match with the list of VAPs the exist in the driver
  - Delete VAP in case the supplied VAPs do not contain the any of VAPs present in the list that comes from the driver

- Modify VAP if there is a mismatch between the VAP parameters supplied and what is presently configured

- Add or delete ACL entry if the ACL list supplied does not match what is presently configured.

□ Prototype

```
a_status_t acfg_set_profile(acfg_wlan_profile_t *profile, acfg_wlan_profile_t *current_profile);
```

□ Parameters

profile: The new profile to be set.

current\_profile: current wlan profile. If the acfg\_set\_profile called for the first time, then this current\_profile would be set to NULL. For any profile modification, current\_profile would provide the current WLAN configuration. acfg\_set\_profile API would do the comparison of new and current profile and does appropriate necessary actions.

■ acfg\_get\_profile

□ Description

This API is used to get the entire current WLAN profile configuration.

□ Prototype

```
a_status_t acfg_get_profile(acfg_wlan_profile_t *profile);
```

□ Parameters

profile: Filled with current WLAN profile configuration on return.

#### 4.3.2.2 ACFG Driver Interface

The ACFG driver interface is implemented under the drivers/wlan/os/linux/src directory. Implementation of ioctl interface handlers and event handlers are described in the following subsections.

##### ioctl interface handler

■ acfg\_handle\_ioctl

□ Description

This is the main acfg ioctl handler function which will be called from

“ath\_ioctl” – for radio interface-related ACFG ioctl commands like creating a VAP or deleting a VAP cfg\_handle\_ioctl, and so on.

“ieee80211\_ioctl” – for VAP interface related ioctl commands.

The ioctl command is ACFG\_PVT\_IOCTL for acfg\_handle\_ioctl to be called.

This routine checks the acfgdispatchentry entry and if it finds valid dispatcher entry in the table, then it calls the appropriate dispatcher function call.

Acfgdispatchentry is an array of \_acfg\_dispatch\_entry which maintains the mapping of ACFG command to corresponding function handlers.

```

typedef struct _acfg_dispatch_entry {
    acfg_cmd_handler_t      cmd_handler;      /* dispatch function */
    uint32_t                cmdid;           /* WMI command to dispatch from */
    uint16_t                flags;
} acfg_dispatch_entry_t;

□ Prototype

int acfg_handle_ioctl(struct net_device *dev, void *data);

```

■ Parameters

*net\_device*: Network interface for radio device  
*data*: Data to be passed to the driver (*ifr->ifr\_data*)

## Event handler

■ *acfg\_send\_event*

□ Description

This routine receives the ACFG events from driver and sends it to the acfg application through the registered Netlink socket. *acfg\_net\_indicate\_event* is the API used to send the events through the netlink socket. *acfg\_send\_event* routine gets called from *osif\_umac.c* for various WLAN driver events such as the following:

- AUTH complete
- ASSOC complete
- DEAUTH complete
- DISASSOC complete
- AUTH indication
- ASSOC indication
- DEAUTH indication
- DISASSOC indication
- Push button

□ Prototype

```

void acfg_send_event(struct net_device *dev, acfg_event_type_t
type,
                     acfg_event_data_t *acfg_event)

```

□ Parameters

*net\_device*: Network interface for Radio device

*type*: ACFG event type

*acfg\_event*: Event data – which is of the format

```

typedef struct acfg_event_data {

```

```

    u_int16_t result;
    u_int16_t reason;
    u_int8_t addr[IEEE80211_ADDR_LEN];
    u_int8_t downlink;
} acfg_event_data_t;

```

## Other WLAN Driver Modifications

The WLAN Driver is mainly modified to get the current WLAN driver configurations. SIOCG80211PROFILE is the ioctl added to get the WLAN driver current configuration for a given VAP.

wlan\_get\_vap\_info is a UMAC API added to get profile information.

### 4.3.3 Configurations

#### 4.3.3.1 Configure AP through ACFG test application

acfg\_tool is the ACFG test application that uses configuration files as input and configures the AP. Each ACFG config file has parameters for a single radio profile. The ACFG config file consists of two blocks:

- Radio parameters
- VAP parameters

Refer to the *Configuration API (ACFG) for 10.2X and 10.4X WLAN Drivers Reference Guide* (80-Y8284-1) for the configuration file template and various supported parameters. After the configuration file is created, run the following command to configure the AP:

acfg\_tool acfg\_set\_profile <new conf file>

Example: **acfg\_tool acfg\_set\_profile /etc/acfg.conf**

After every set\_profile command, the current configuration for the radio is stored in **/etc/acfg\_cur\_wifiX.conf (X=0/1)**.

#### Enable bridge configuration

To enable bridge configuration:

- Create bridge if it is not present.
- Enable bridge config in hostapd config file [bridge=br0].
- Run the hostapd.
- Run the supplicant with -b <bridge\_name> option.  
**wpa\_supplicant -b <bridge\_name> -g /var/run/wpa\_supplicant/global &**
- Run the acfg\_tool.
- Add the created VAPs to bridge.

## Create VAPs

Edit the acfg configuration file to fill the radio and VAP parameters and execute the acfg\_tool command

```
acfg_tool acfg_set_profile /etc/acfg.conf
```

If the profile configuration was for wifi0, the current configuration will be saved in /etc/acfg\_cur\_wifi0.conf

## Delete VAPs

To delete one or more VAPs, remove the VAP entry from acfg configuration file.

VAP entry will be starting from “vap\_name” till all the VAP specific parameters in the configuration file.

## Modify VAP parameters

Change the VAP parameters and execute the acfg\_tool command

```
acfg_tool acfg_set_profile /etc/acfg.conf
```

## Creating profile on multiple radios

An ACFG configuration file corresponds to a radio profile. To create VAPs on a dual radio configuration, the acfg\_tool command has to execute twice, each time with single radio profile.

To create a profile on wifi0, the “radio\_name” parameter should be assigned as “wifi0”. All the following VAP parameters in this file will be attached to this radio. Execute the acfg\_tool command to create VAPs on wifi0.

```
acfg_tool acfg_set_profile /etc/acfg_wifi0.conf
```

To create a profile on wifi1, the “radio\_name” parameter should be assigned as “wifi1”. Execute the acfg\_tool command to create VAPs on wifi1.

```
acfg_tool acfg_set_profile /etc/acfg_wifi1.conf
```

## GET profile

To get a profile, use the following command.:

```
acfg_tool acfg_get_profile wifiX
```

This will retrieve the complete profile information of a radio.

### 4.3.3.2 Configuring WPS, Radius authentication, VLAN and WDS on reference board

#### WPS CONFIGURATION

The following parameters must be added to the acfg configuration file for WPS:

```
wps_upnp_iface=br0
wps_device_name=AtherosAP
wps_flag=1
```

The following commands must be executed on the reference board for push button, pin method, and JumpStart functions, and AP/STATION VAP should be created before executing these commands:

- PUSH BUTTON: “acfg\_tool acfg\_wps\_pbc vap0”

After executing the preceding command on the board (if board is configured as AP), press the push button on the wireless station within two minutes for successful association.

- PIN METHOD: “acfg\_tool acfg\_wps\_pin vap0 set 12345670 0” (12345670 is the WPS pin and WPS pin should be valid)

After executing the above command on the board (if board is configured as AP), use the same WPS pin on the wireless station. The association should be completed within 5 minutes.

- JumpStart

Press the PBC button on both the AP and station side. (The PBC button must be pressed on host board for offload).

#### RADIUS CONFIGURATION

The following parameters should be added for radius authentication:

```
radius_ip=xxx.xxx.xxx.xxx
radius_port=1812
radius_sharedsecret=xxxxxxxx
```

The radius port and shared secret should be the same as in the radius server configuration.

```
authtype=6
cipher=1 for WPA authentication
authtype=7
cipher=2 for WPA2 authentication
```

#### WDS CONFIGURATION

The following parameters need to be added in the configuration file to create ROOT AP/Repeater AP/Extended AP

- ROOT AP AND REPEATER AP

Root AP (Reference board #1):

    Add wds\_enabled=1 in the configuration

Repeater AP (Reference board #2):

    Add wds\_enabled=1 in the configuration

Configure Station VAP first

- ROOT AP AND REPEATER INDEPENDENT AP

Root AP (Reference board#1):

    Add wds\_enabled=1 in the configuration

Repeater INDEPENDENT AP (Reference board #2):

    Add wds\_enabled=1and wds\_flag=1 (for both AP and station AP) in the configuration

    Configure Station VAP first

- ROOT AP AND Extended AP

Root AP (Reference board #1):

    WDS related configuration parameters are not required to be added in configuration file.

Extended AP (Reference board #2):

    Add wds\_flag=2 (for station AP) in the configuration

    Configure station first

- ROOT AP AND Extended STA

Root AP (Reference board #1):

    WDS related configuration parameters are not required to be added in the configuration file.

Extended AP (Reference board #2):

    Add wds\_flag=2 (for station AP) in the configuration file.

    Only the station VAP should be configured in the reference AP

- ROOT AP MAC

Station will associate to the AP using the MAC address configured in station

Root AP (Reference board #1):

    Add wds\_enabled=1 in the configuration

Repeater AP (Reference board #2):

    Add wds\_enabled=1 in the configuration

    Add wds\_addr=xx:xx:xx:xx:xx:xx (root AP mac address)

    Configure Station VAP first

## VLAN CONFIGURATION

The following parameters need to be added to the configuration file:

vlanid=2

(The vlanid should be configured in VLAN switch)

## Events

The events file (acfg\_event\_log) will be stored under /etc folder. This file will be removed after reboot.

QUALCOMM®  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

#### 4.3.3.3 Acfg configuration file: Template and Parameters

```
#Radio parameters
#There can be only one radio_name entry in a file radio_name=wifi0
#ieee channel number

channel=6 countrycode=842

#VAP parameters starts here vap_name=vap0
#opmode values
#adhoc=0
#sta=1
#wds=2
#ahdemo=3
#ap=6
#monitor=8 opmode=6

#phy mode values
#auto=0
#11a= 1
#11b=2
#11g=3
#FH=4
#11na ht20 =5
#11ng ht20 =6
#11na ht40plus=7
#11na ht40minus= 8
#11ng ht40plus= 9
#11ng ht40minus=10 phymode=3 ssid=tintin bitrate=54 beaconint=100
txpower=20 rtsthresh= fragthresh= vapmac=
#authmode values
#open=0
#wep=1
#wpa=2
#wpa2=3
#wps=4 authtype=2

#cipher values
#tkip=1
#aes=2

cipher=3
#passphrase psk=secretpassword
#wep key index0 wep_key0=1234567890
#wep key index1 wep_key1=1111111111
#wep key index2 wep_key2=2222222222
#wep key index3 wep_key3=3333333333

wps_flag= wps_upnp_iface= w ps_device_name= wps_pin= wps_init_handshake=
eap_type= eap_identity= eap_password= ca_cert_path= client_cert_path=
private_key_path= private_key_password= radius_ip= radius_port= radius_
sharedsecret=
```

```

#acl mac list
acl_node=00:11:22:33:44:55 00:12:13:14:15:16
acl_node_sec=11:22:33:44:55:66 12:23:34:45:56:67
#acl policies
#allow=1
#deny=2
acl_policy=0
acl_policy_sec=0
wds_enabled= wds_addr=00:11:22:33:44:55 vap_ind=
extap=
#vap_name=vap1
.
.
.

#vap_name=vap2
.
.
.
```

#### 4.3.3.4 ACFG maximum clients event

|                         |                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Synopsis</b>         | An ACFG event is sent to user space, when number of Wi-Fi clients exceeds the maximum client configuration of AP.                                                                                                                                                                                                                                                                     |
| <b>Usage Guidelines</b> | When a new Wi-Fi client associates, ol_ath_node_alloc() is called to allocate new node resource in host and firmware. A check is made to determine whether the number of total WIFI clients exceeds the AP's max supported number of clients. If the check is true, send an acfg event.                                                                                               |
| <b>Usage Example</b>    | <p>Setup AP</p> <p>Start acfg event detection on AP</p> <pre>acfg_tool -e wifi0&amp;</pre> <p>Associate more than 513 clients OR reduce max supported clients to smaller number for verification by passing qca_ol module parameter, e.g. max_peers=400</p> <p>Check the acfg log file</p> <pre>root@OpenWrt:/# tail -f /etc/acfg_event_log wifi0: ACFG Event exceed Max client</pre> |

# 5 WLAN AP Modes

---

This chapter describes the different modes or variants of operation of WLAN APs, such as AP mode, client mode and security, multiple basic service set identifier (BSSID), virtual local area network (VLAN) support, Hy-Fi WLAN, AP-only mode, Positioning: LOWI on AP, Location wireless interface (LOWI).

## 5.1 AP Mode

The AP mode is the standard mode of operation that provides all the basic functionalities of an access point in an infrastructure BSS. These functionalities include sending beacons to advertise the BSS identity and capabilities and managing associations with clients in the infrastructure network. This section describes the software implementation for basic AP mode operation. Refer to other sections for advanced features in AP mode.

The following are the basic functions in AP mode:

- Sending beacons and updating the information sent in beacons based on various changes in the environment
- Allowing association of clients through basic 802.11 authentication/association handshake
- Data forwarding, which includes forwarding data between clients and the distribution system
- Buffering and delivering unicast frames to the clients that are in power save mode and delivering multicast/broadcast data frames after advertising the presence of data through beacons

### 5.1.1 Terminology

**VAP (Virtual AP):** The software entity that represents the AP. The term ‘virtual’ is used because multiple logical AP entities can be present on a single physical device. Refer to the ‘MBSSID’ section for more information.

**Node:** The software structure that represents an AP or the association station. In AP mode, the Tx/Rx capabilities and current state of each client station in the BSS is maintained in the node corresponding to the client and is used to exchange data with the client.

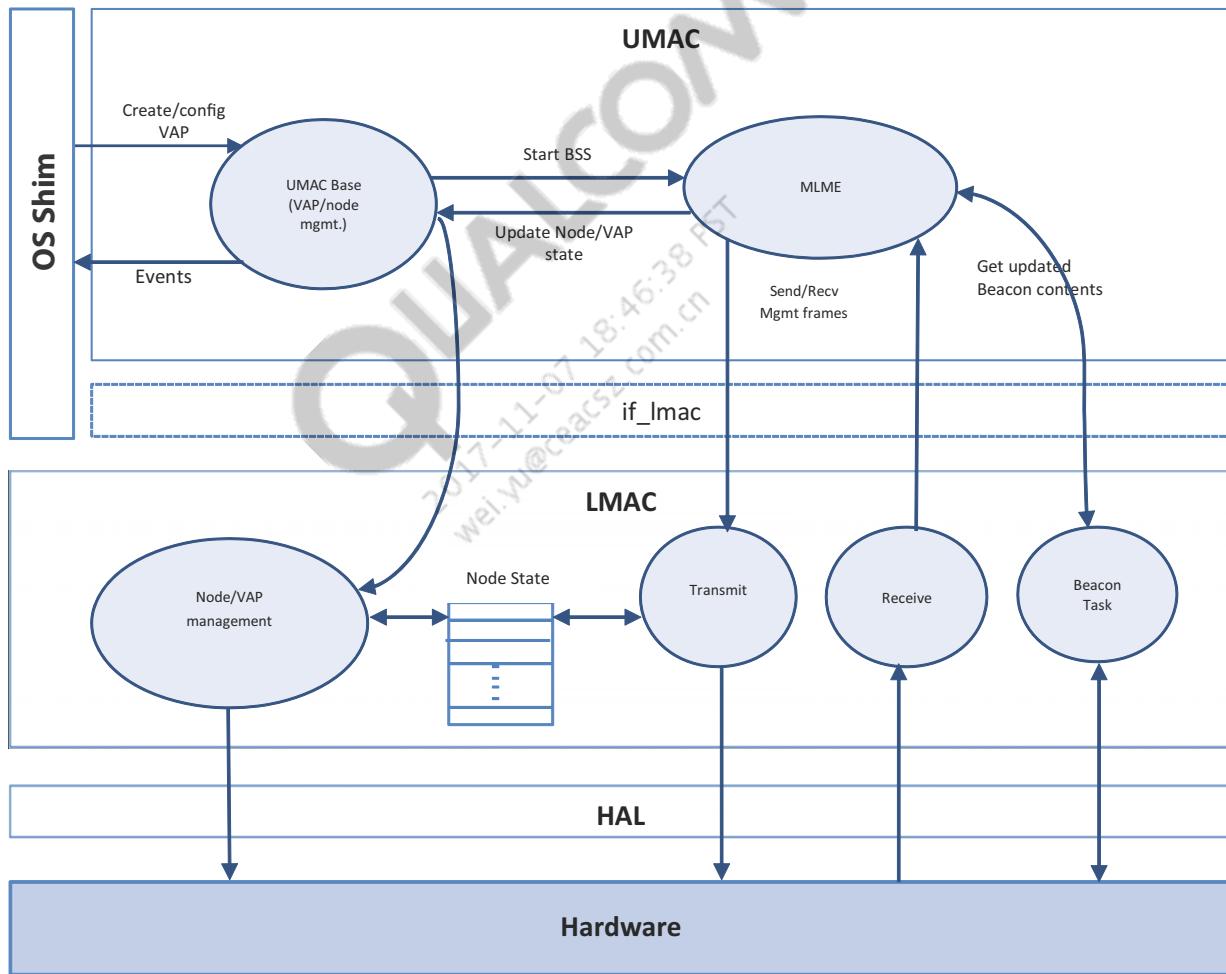
### 5.1.2 Implementation

The implementation of basic AP functionality is distributed across many modules. This section describes the implementation details and the major parts of the code implementing the basic functions of AP mode operation. The following subsections describe key functions. The overall

dependencies of these functions in the software architecture are illustrated in [Figure 5-1](#). Parts of codebase corresponding to each of the functions shown in [Figure 5-1](#) are explained in the subsections.

### 5.1.2.1 Full Offload Architecture

In this mode, the WLAN driver components and the hostapd/supplicant from the WLAN application component runs on the target and the remaining application sub components of WLAN application run on the host. A thin target interface layer is added on the host and a thin host interface layer is added on the target for host-target communications. The hardware interface between the host and target could be a USB or MII interface.



**Figure 5-1 AP Mode Operation**

### 5.1.2.2 BSS Management

The major functions for BSS management are 1) the creation of an infrastructure BSS network using the parameters chosen by the user, such as ESSID, channel (auto channel selection is invoked if the channel is not fixed, which is explained in a separate section), and channel mode; 2) management of VAP for the access point; and 3) managing the clients associated to the AP.

#### **umac/base/ieee80211\_vap.c**

The main API in this file is *wlan\_vap\_create*, which is used by the OS shim layer to create an AP interface. This is a generic API for creating a VAP and the AP specific path is chosen by invoking this API with ‘opmode’ set to IEEE80211\_M\_HOSTAP. This file has basic VAP management interfaces for the OS shim layer. It includes interfaces for the creation and deletion of VAP, for managing event handlers for various VAP-related events, and for retrieving VAP state information such as statistics. It also contains VAP setup functions used by lower layers (if\_lmac). The VAP management interfaces are generic for different possible modes of operation such as ad hoc mode and client mode.

#### **umac/base/ieee80211\_config.c**

This code implements the basic configuration API. *wlan\_set\_desired\_ssidlist* is used to setup the SSID for the AP. *wlan\_set\_param* and *wlan\_get\_param* are used to set and retrieve other basic parameters like beacon interval, fragmentation threshold, RTS threshold, and so on.

#### **umac/base/ieee80211\_channel.c**

This code implements the channel related configuration API, which includes *wlan\_set\_channel* used to set the desired channel, *wlan\_set\_desired\_phymode* to setup the operating phy mode of AP.

The main API in this file for AP mode operation is *wlan\_mlme\_start\_bss*, which will start the AP with the given SSID and channel already configured through the configuration interfaces *wlan\_set\_channel* and *wlan\_set\_desired\_ssidlist*.

#### **umac/base/ieee80211\_node\_ap.c**

A node is a software structure which represents a station (AP or client) and maintains the state and capability information of the AP for each client associated to the AP. This file consists of node management functions specific to AP functionality, which includes AP node creation, management of client nodes when a client joins or leaves the infrastructure network. Generic node management functions used here are implemented in the file umac/base/ieee80211\_node.c. This file also implements the node inactivity timer to ensure that stale nodes are removed from the BSS.

#### **umac/mlme/ieee80211\_mlme\_ap.c**

This file contains the implementation of MAC layer management entity for the AP mode. This includes the creation of the infrastructure network by setting up the chosen ESSID and channel and the processing of authentication/association requests from clients. *mlme\_recv\_auth\_ap* processes and responds to authentication requests from the client. *ieee80211\_mlme\_recv\_assoc\_request* processes and responds to association requests from clients. Basic capabilities like supported rates advertised by clients are updated in the corresponding nodes using nodes in the UMAC and LMAC layers, which are later used by transmit and rate control modules.

**umac/mlme/ieee80211\_mgmt\_bss.c**

This file implements processing of probe requests received from clients and sending probe response frames.

**umac/mlme/ieee80211\_mgmt\_ap.c**

This file implements the actual parsing of management frames received by AP and also setting up the management frames to be transmitted by AP.

**umac/mlme/ieee80211\_proto.c**

This file implements the generic state machine for VAP. The major states in AP mode are:

- INIT is entered when VAP is created or reinitialized
- SCAN is entered only if auto channel selection is required
- JOIN is entered to wake up the chip and to setup the frequency of operation and receive filters in hardware to receive frames that are appropriate in AP mode
- RUN is the normal running state with beacon transmission started

### 5.1.2.3 Beacon Generation

Periodic beacon generation is done with the help of hardware timers that can be programmed based on the beacon interval configured by user. A timer called software beacon alert (SWBA) is used to 1) generate a hardware interrupt at a programmable time before each TBTT; 2) allow the software to form an updated beacon frame based on the current state; and 3) queue the frame into the hardware queue for transmission. The actual DMA of the frame occurs based on another programmable timer called DMA Beacon Alert (DBA). Implementation of beacon generation is split between LMAC and UMAC layers in the source files list below:

**umac/mlme/ieee80211\_beacon.c**

This file contains the code to 1) allocate and set up the contents of beacon frames based on various AP capabilities; and 2) dynamically update the beacon contents based on changes in AP state. The code that will actually populate the information elements is present in *umac/mlme/ieee80211\_ie.c*.

**lmac/ath\_dev/ath\_beacon.c**

This file contains lower level code for beacon generation: setting up beacon timers, configuring the hardware queue used to send beacons, and setting up descriptors based on the packet buffer created by upper layers. This file contains the implementation for the SWBA interrupt handler. It accesses the UMAC functions to form beacon frames through the *if\_lmac* layer. This code also contains the implementation for buffered multicast/broadcast frame delivery, which includes setting the DTIM in the beacon and queuing the multicast frames to the hardware queue reserved for this purpose, so that the frames follow immediately after beacon transmission.

### 5.1.2.4 Power Management

#### **umac/pm/ieee80211\_power\_queue.c**

This file implements the power save queue used to queue frames destined to clients in the legacy power save mode. Frames will be queued during transmission using *ieee80211\_node\_saveq\_queue* if the destination client is in power save mode as indicated by the node representing the client, and will be sent out of the queue using *ieee80211\_node\_saveq\_send* when a PS-Poll frame is received from the client.

#### **umac/pm/ieee80211\_ap\_power.c**

Functions related to power management such as setting up a TIM to be updated in subsequent beacon transmissions are implanted in this file.

### 5.1.2.5 Data Forwarding

Data forwarding occurs when the AP forwards data between the associated clients in the BSS and the distribution system to which the AP is connected.

#### **umac/txrx/ieee80211\_output.c**

This file implements generic transmit functionality. The OS shim layer uses the interface *wlan\_vap\_send* to transmit data frames. If the operating mode is AP, the destination address is checked that it is of an associated client. The client node is retrieved from the node table to get the node information such as power save state, whether unicast key encryption is enabled, and rate/phy capabilities, which are required to send the frame. Since the wireless interface is exposed as an Ethernet interface to the upper layers, data to be transmitted to the wireless network will be in the form of Ethernet packets. These packets are translated to 802.11 frames after checking if the destination is part of the BSS. Packet classification into one of the WMM access categories is also done here. Two checks are performed: 1) whether the destination client is in power save mode and 2) whether the buffers the packet are in power save queue if required.

#### **umac/txrx/ieee80211\_input.c**

This file implements receive functionality for all modes. In AP mode, the source address of the received packet is verified and if is not from an associated client, the packet is discarded and a disassociation request is sent to the station. Intra-BSS bridging is also done here by checking if the received data packet is destined for a client with the same BSS and if so, it is forwarded to the transmit module without indicating the packet to upper layers. Multicast/Broadcast packets are forwarded to both upper layers as well as to the other stations in BSS.

### 5.1.2.6 Configuration

The following command is used in Linux to create an AP interface:

```
wlanconfig ath create wlandev wifi0 wlanmode ap
```

The basic configuration required for an AP mode operation can be configured using the following command:

```
iwconfig ath0 essid "<AP_SSID>" mode master freq <channel>
```

**NOTE** Client disconnects and reconnects with the AP continuously after an hour of streaming traffic, due to continuous disassociation. The Internet browsing speed considerably

reduces in the dissociated client, while other associated clients do not experience any reduction in speed. Based on sniffer analysis and third party APs, this problem appears to be client-specific.

**NOTE** An SCTP association from second LAN client to same WAN server is not established, when an outbound SCTP association from two LAN clients to the same WAN server are initiated and both the SCTP connections are attempted to be terminated from the LAN clients. Two LAN clients cannot establish simultaneous SCTP connections to same server. This behavior occurs only on QCA9531, QCA9558, and QCA9563 platforms when the kernel version used is 3.3.8.

**NOTE** No response from FTP PASV command is received on WAN when an outbound TCP connection from an AP to an FTP server port is established and PASV command is sent. Customer cannot make passive FTP connections from AP to the FTP server. This behavior occurs only on QCA9531, QCA9558, and QCA9563 platforms on which the kernel version is old.

## 5.2 AP-only Mode

### 5.2.1 Data Path Optimizations For Improved CPU Utilization

Various optimization techniques that have been implemented inside the WLAN driver data path to minimize CPU utilization (and thereby achieve overall improved throughput performance). In the following subsections, the details of the APONLY code are presented, which yields significant reduction in CPU utilization along with improved throughput performance. The APONLY support can be enabled at compile time in the board-specific configuration file using “export UMAC\_SUPPORT\_APONLY=1”. For example, for the db120 case, the configuration file would be “build/scripts/db12x/config-db12x”.

#### 5.2.1.1 APONLY Support

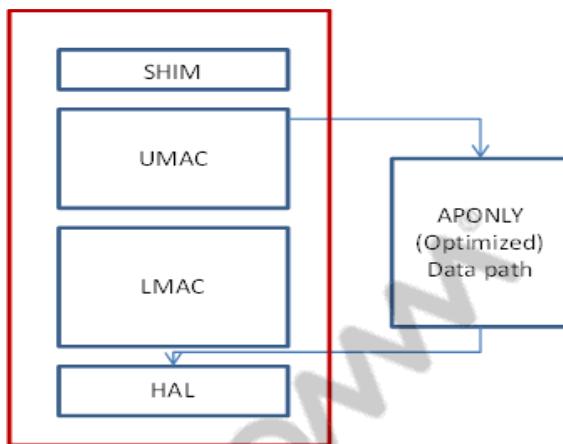
The data path code contributes significantly to the overall CPU utilization of the system and it also plays a crucial role in the final throughput performance. Therefore, optimizing the data path code helps in yielding improved performance.

The profile data collected on various platforms to analyze the data path for potential optimization opportunities has shown that there are significant I-cache and D-Cache misses incurred, and also, the number of instructions executed per packet is quite high for both the tx and rx paths. Motivated by these observations, the APONLY mode was created to re-organize the data-path code with the goal of reducing the I-cache, D-cache misses and the total number of instructions required to send/receive a packet. In the following we present the details of the APONLY support implementation.

#### 5.2.1.2 APONLY Implementation

The APONLY code is an additional code path added to the existing code. This additional code path is built into the WLAN driver when the compile flag UMAC\_SUPPORT\_APONLY is enabled in

the build configuration. Figure 5-2 shows the overall data path architecture with the APONLY code.



**Figure 5-2 APONLY Data Path**

### APONLY optimizations

The APONLY code is created as a function-by-function copy of the original data path code (functions taken from shim, UMAC and LMAC) to start with, and then various optimization techniques have been applied on these individual functions in the aponly code path. Therefore, aponly code is an optimized version of the original data path code. The copied functions are renamed with a common suffix, ‘\_aponly’. For example, the ‘wlan\_vap\_send()’ function in the aponly code is named as ‘wlan\_vap\_send\_aponly()’. The entire aponly code is organized in three files:

- drivers/wlan/os/linux/src/ieee80211\_aponly.c – majority of the function definitions.
- drivers/wlan/os/linux/include/ieee80211\_aponly.h – aponly switch points (described later).
- drivers/wlan/lmac/ath\_dev/include/ath\_aponly.c – export symbols from LMAC to aponly code.

The following optimizations have been made in the aponly code path:

1. Move all (or majority of) the data path functions to one single file (ieee80211\_aponly.c):

#### Advantages

- All the relevant functions lay close by in the text section (and in the memory layout) – enhances spatial locality.
- This also avoids function pointers, and now functions can be invoked directly as the related functions are all in one file – reduces overhead in terms of cycles and also reduces the I-cache misses.
- Functions can now be inlined as they now reside in the same file. Function inlining sometimes offers significant performance improvements as described in the next optimization item.

#### Disadvantages

- The flexibility of layering – the functions pulled into ieee80211\_aponly.c – actually belong to different layers (shim, UMAC, and LMAC).
- Also, since aponly functions can call procedures in other modules directly without any function pointers, the other modules (for example, ath\_dev) should now export these symbols.

## 2. Function inlining

Function inlining helps in more than one way. Firstly, it reduces the function call overhead and it also offers increased opportunity to the compiler to perform optimizations like copy propagation, constraint propagation, better register allocation, and so on. Of course, inline functions can sometimes be inefficient too. The compiler executes its discretion to determine which of the “inline” marked functions to actually include inline in the final object code.

## 3. Function level optimizations/changes:

- Delete station, wds, nawds, code.
- Rearrange code to enhance spatial locality (access one data structure and finish with it when possible, and move on to working with other data structure – good for D-cache hits).
- Add likely/unlikely functions – to reduce branch mis-predictions.

## APONLY code entry points

The following table lists the aponly code entry points along with the corresponding regular/generic code entry points. It is through the invocation of these functions that the driver enters into the aponly code. The definitions of these functions are located in the ieee80211\_aponly.c file.

| aponly function name          | Corresponding generic function name |
|-------------------------------|-------------------------------------|
| osif_vap_hardstart_aponly()   | osif_vap_hardstart_generic()        |
| ath_netdev_hardstart_aponly() | ath_netdev_hardstart_generic()      |
| ath_handle_intr_aponly()      | ath_handle_intr_generic()           |
| ath_intr_aponly()             | ath_intr_generic()                  |
| ath_isr_aponly()              | ath_isr_generic()                   |

## APONLY run-time switch points

As the APONLY code path does not support certain features and modes, the driver cannot provide such features when running in the APONLY mode (that is, when UMAC\_SUPPORT\_APONLY=1). To address this issue to a certain extent, the run-time switch points are created that will check the current VAP mode and invoke aponly code or the generic code dynamically. Please note that this method cannot be used to support every feature not present in the aponly code.

[Table 5-1](#) describes the two switch points (both of which are located in drivers/wlan/os/linux/src/ieee80211\_aponly.h):

**Table 5-1 APONLY Switch Points**

|                            |                                                                                                                                                                                                                                                                                |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| umac run-time switch point | Invoke aponly UMAC entry point if all of the following conditions are true:<br><ul style="list-style-type: none"> <li>■ VAP is in HOSTAP_MODE</li> <li>■ VAP NAWDS mode is disabled</li> <li>■ VAP WDS mode is disabled</li> </ul> Otherwise, invoke generic UMAC entry point. |
| lmac run-time switch point | Invoke aponly LMAC entry point if all of the following conditions are true:<br><ul style="list-style-type: none"> <li>■ VAP is in HOSTAP_MODE</li> <li>■ VAP nawds mode is disabled</li> <li>■ VAP WDS mode is disabled</li> </ul> Otherwise, invoke generic LMAC entry point. |

### LMAC symbol exports

The APONLY code is currently built as a part of the Direct-Attach binary (qca\_da.ko). Since the APONLY code does not use function pointers to call into LMAC functions due to efficiency reasons, the LMAC function symbols should be made available so that the aponly code will load properly. Therefore all the symbols in LMAC that are referred by the aponly code are exported using EXPORT\_SYMBOL in the file drivers/wlan/lmac/ath\_dev/ath\_aponly.c.

#### 5.2.1.3 APONLY support – Frequently Asked Questions (FAQs)

- When should APONLY mode be used?

The APONLY mode should be used when the CPU performance critical is a critical factor and when the features not-supported by APONLY are not needed.

- Which features works when UMAC\_SUPPORT\_APONLY is enabled?

The following is a list of modes/features that have been tested and verified to work with aponly mode enabled.:.

- Security
- WDS
- MBSSID
- AP-Switch
- Extender AP
- VoW
- 802.11n Wi-Fi
- WPS
- Multicast enhancements
- Regulatory
- NAWDS

## 5.3 Client Mode

The client (STA) mode of operation is supported in an AP platform to enable wireless bridging applications like WDS. The basic requirement is to associate to an infrastructure network selected by user and manage the connection to the AP to be able send and receive data from the network. This section describes the software implementation of the basic client mode operation.

Following are the basic functions in client mode:

1. Connection management which includes the following functions:
  - a. Scanning the wireless channels to find the infrastructure networks (APs) available
  - b. Establishing connection to an AP using the authentication/association handshake and managing the connection to ensure continuous connectivity and compliance to any dynamic changes to the BSS as indicated by the AP through beacons
2. Data exchange with the distribution system through the AP
3. Power management

### 5.3.1 Terminology

VAP (Virtual AP): The software structure that represents the client interface. The term ‘virtual’ is used because multiple logical AP or client entities can be present on a single physical device.

### 5.3.2 Implementation

The implementation of client functionality is distributed across many modules. This section describes the implementation details and the code base implementing the basic functions of client mode operation. The client mode-specific functionality is implemented mostly in the UMAC layer and uses the generic low-level functions from the LMAC. The description is further divided into subsections based on the key functions. The overall dependencies of these functions in the software architecture are illustrated in [Figure 5-3](#). The key generic modules used by client mode operation include scanning, connection management, transmit/receive and VAP management. Parts of the code base corresponding to each of the functions shown in [Figure 5-3](#) are explained in the subsections that follow.

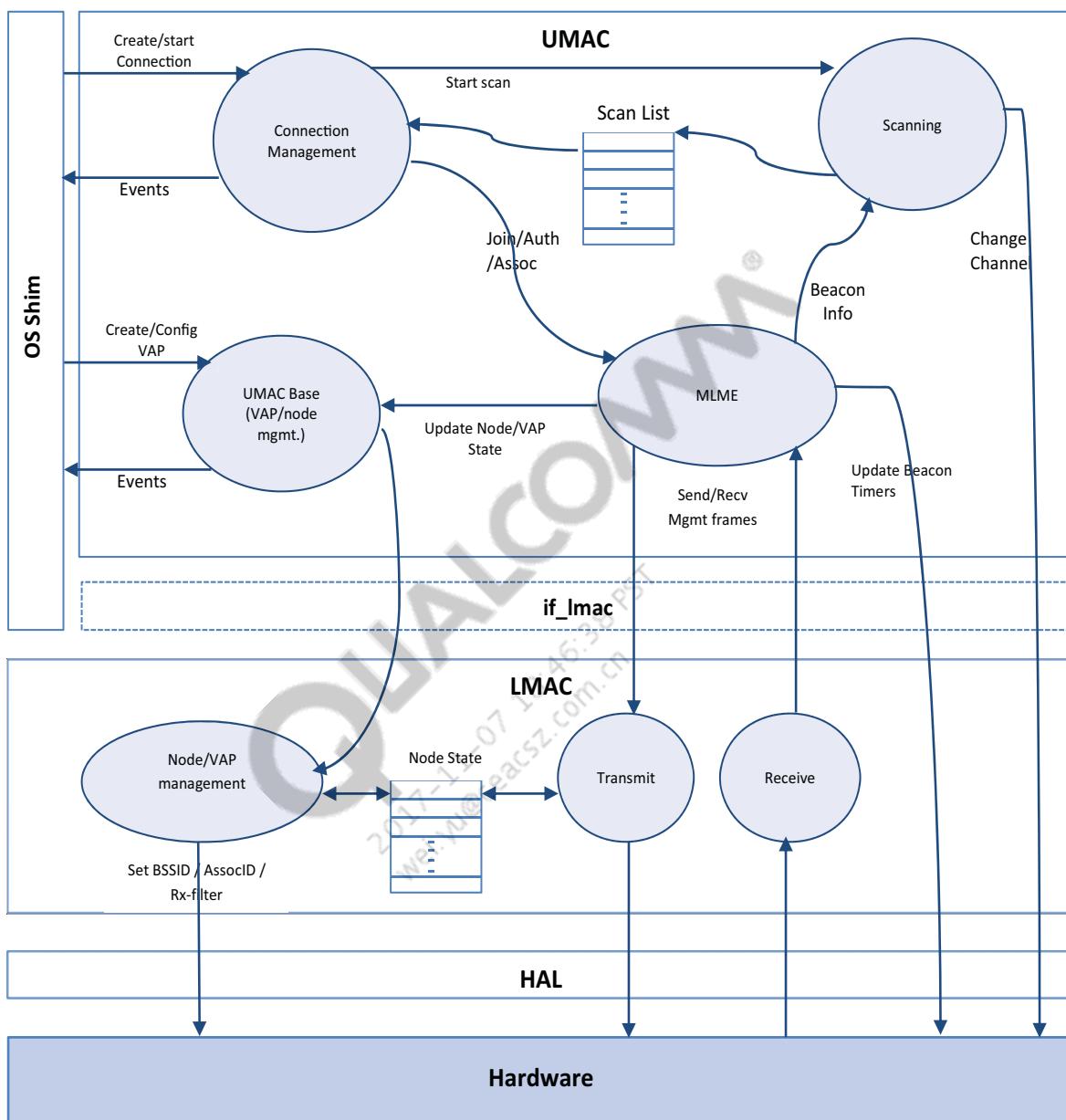


Figure 5-3 Client Mode Operation

### 5.3.2.1 VAP interface creation and configuration in client mode

The major functions are creating a client VAP and setting a basic configuration like a desired SSID.

#### **umac/base/ieee80211\_vap.c**

The main API in this file is *wlan\_vap\_create*, which is used by the OS shim layer to create a client interface. This is a generic API for creating a VAP and the client-specific path is chosen by invoking this API with ‘opmode’ set to IEEE80211\_M\_STA. This file has basic VAP management interfaces for the OS shim layer. It includes interfaces for 1) the creation and deletion of VAPs, 2) managing event handlers for various VAP-related events, and 3) retrieving VAP state information such as statistics. It also contains VAP setup functions used by lower layers (if\_lmac).

*umac/base/ieee80211\_config.c* implements the basic configuration API. *wlan\_set\_desired\_ssidlist* is used to set up the SSID of the BSS that the client desires to join. *wlan\_set\_param* and *wlan\_get\_param* are used to set and get basic wireless parameters.

### 5.3.2.2 Connection Management

This section describes the implementation of connection establishment and management for client mode operation. The generic ‘Connection Management’ framework is used for establishing and maintaining connection with the AP in an infrastructure network.

The connection state machine for the client mode is created from the OS shim layer using the API *wlan\_connection\_sm\_create* while creating the client interface. A connection event handler should be registered with the connection manager to get connect/disconnect events that are used to indicate link state changes to the OS.

The connection state machine is started by the OS shim layer while starting the client interface using the API *wlan\_connection\_sm\_start*. The connection manager starts with a scan operation if there is no valid scanlist already available.

*umac/scan/ieee80211\_scan.c* implements the scanning task, which is started by the connection manager using the API *wlan\_scan\_start*. The client scans to find the infrastructure APs available in all the allowed channels. The list of channels to be scanned is created by *ieee80211\_scan\_update\_channel\_list* during the initialization of the generic scanner task (*ieee80211\_scan\_attach*). This list is actually derived from a list of allowed channels prepared by the HAL module based on the device capabilities (bands of operation supported) and regulatory domain configured. The client scans by switching to each channel and collects the AP information from the beacons received. The available infrastructure networks (APs) are stored in the scan table. From this scan table, a candidate AP list is built (*umac/scan/ieee80211\_aplist.c*) that consists of those APs with SSIDs matching the desired SSIDs chosen by the user.

The connection manager enters the CONNECTING state once the scan is completed. In this state, the client tries to associate with each of the APs in the candidate list built by scanner module. *wlan\_assoc\_sm\_start* (*umac/sme/ieee80211\_assoc\_sm.c*) starts the association state machine which goes through the join (probe)/authentication/association sequence.

*umac/mlme/ieee80211\_mlme\_sta.c* implements the operations required by association state machine. The join operation involves setting the channel corresponding to the scan entry and sending a probe request and processing a probe response.

This is following with an authentication and association frame exchange with the selected AP. Processing of an association response includes setting up the rates that can be used for transmitting data frames to the AP. If the association fails and if there are more APs in the candidate AP list, the process is repeated with next AP in the list.

On successful completion of association to the selected BSS, the hardware is setup with the association ID allocated to the client and the BSSID using *ath\_hal\_setassocid* (from *ath\_vap\_up* in *lmac/ath\_dev/ath\_main.c*). This enables the hardware to receive data frames from the AP and also allows the hardware to wake-up on TIM indications in beacons.

The connection state machine enters the connected state, whereupon a timer is started to check periodically whether roaming is to be done. If the RSSI of the current AP falls below a threshold (default is 15 dBm), the candidate AP list is rebuilt from the scan list to determine if there is any other AP; if another is found, roaming is initiated by entering the disconnected state and restarting the association state machine with the scan entry corresponding to the new AP. A disconnect is also triggered if beacons are missed for a configurable number of consecutive beacons from the AP (the default is 15 beacons).

### 5.3.2.3 Power Management

The client mode power management function saves power by putting the device in sleep mode when there is no activity for a specific duration, which is configurable by the user based on trade-off between throughput and power consumption.

*umac/pm/ieee80211\_sta\_power.c* implements the client mode power save functionality as a state machine, which is initialized by *ieee80211\_sta\_power\_vattach* during VAP initialization (from *ieee80211\_vap\_setup*). When the client starts running (after associating to the AP), it enters the ACTIVE state and a timer is started to monitor the traffic activity. The activity timer handler periodically checks the time elapsed since the last frame was transmitted or received from the AP (*ieee80211\_sta\_power\_activity\_timer*). The station enters the power save state if this duration is more than the inactivity duration configured. On entering the power save state, the device is put into network sleep mode and the state change is indicated to the AP so that the AP will queue the packets destined to this client while in power save mode.

*lmac/ath\_dev/ath\_power.c* implements device level functions that are used by UMAC to change the power save state of the chip. *ath\_pwrsave\_set\_state\_sync* uses the HAL interface *ath\_hal\_setpower* to put the device in the sleep or awake state.

When the device is in network sleep mode, it wakes up periodically based on beacon interval timers programmed in client mode to check TIM fields in beacons received from the AP. If there is data meant for the client, the AP sets the TIM bit corresponding to this client in the beacon frame, which is then indicated to the upper layer (power save state machine). On receiving a TIM/DTIM indication, the power save state machine wakes up the chip and sends a null data frame to the AP, with the power save bit cleared in the frame control field so that the AP can send the frames queued for this client.

### 5.3.2.4 Data Exchange

In client mode, data frames are always sent and received to and from the distribution system through the AP.

`umac/txrx/ieee80211_output.c` implements a generic transmit functionality. The OS shim layer uses the interface `wlan_vap_send` to transmit data frames. In client mode operation, the node used for transmission is always the BSS node corresponding to the AP. The node structure consists of the MAC address of the AP, a unicast key if encryption is enabled and rate/PHY capabilities, which are required to send the frame. Since the wireless interface is exposed as an Ethernet interface to the upper layers, data to be transmitted to the wireless network will be in the form of Ethernet packets. These packets are translated to 802.11 frames. Classification of packets into one of the WMM access categories is also done here.

`umac/txrx/ieee80211_input.c` implements a generic receive functionality. In client mode, the reception of unicast and multicast data frames will be indicated to the power-save state machine.

### 5.3.3 Configuration

The following command is used in Linux to create a client interface:

```
wlanconfig ath create wlandev wifi0 wlanmode sta nosbeacon
```

A basic configuration required for an AP mode operation can be accomplished using the following command:

```
iwconfig ath0 essid "<AP_SSID>" mode managed
```

## 5.4 AP and Client Security

All the security protocol aspects in the system are handled by the hostap application. Hostap applications are based on the open source hostap project and consists of two major applications: hostapd and wpa\_supplicant.

- **hostapd** is a user space daemon for access points, including, for example, IEEE 802.1X/WPA/EAP authenticator for the number of Linux and BSD drivers, RADIUS client, integrated EAP server, and RADIUS authentication server.
- **wpa\_supplicant** is a user space IEEE 802.1X/WPA supplicant for wireless clients for a number of Linux, BSD, and Windows drivers

The WLAN driver provides all necessary interface support and security services for these applications. The scope of this section is limited to explaining those WLAN driver support only.

Refer to <http://hostap.epitest.fi/> for detailed documentation on hostapd/wpa\_supplicant and all the features supported by these applications.

### 5.4.1 Theory of Operation

hostapd/wpa\_supplicant has different driver interface modules to support different WLAN drivers. In this AP platform, hostapd uses the `driver_atheros` interface module and wpa\_supplicant uses the `driver_athr` interface module.

These driver interface modules communicate with the WLAN driver through the IOCTL and wireless events interfaces. The modules also communicate with the network stack through sockets to send and receive data packets for the WLAN network interface.

The WLAN driver provides the required security services for the hostap application, which are accessed through IOCTL interface.

#### 5.4.1.1 Driver Interface for Hostap Application

Figure 5-4 describes the different interfaces provided by the driver to hostapd/wpa\_supplicant.

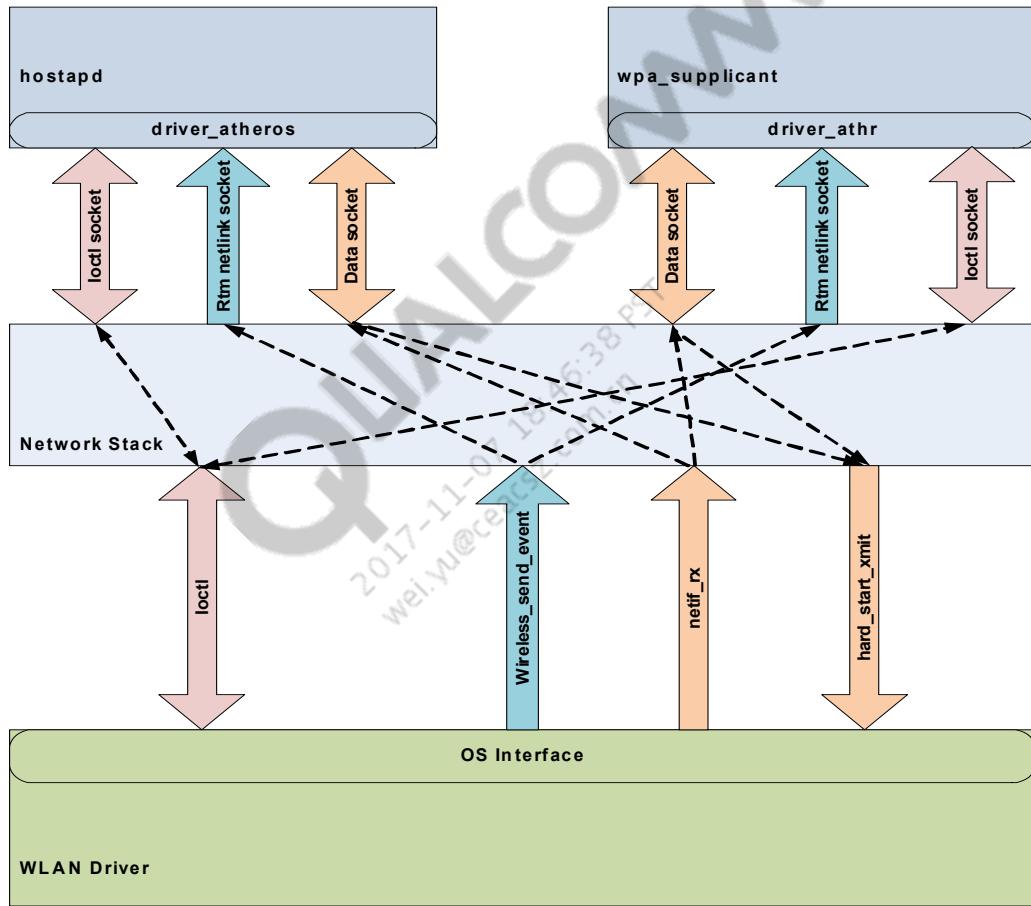


Figure 5-4 Driver Interface for Hostap

#### 5.4.1.2 IOCTL interface

The hostap application opens a datagram socket interface and uses this socket for all ioctl calls to the driver. These ioctl calls get mapped to the WLAN driver ioctl function call which is registered to the WLAN network interface.

### 5.4.1.3 Wireless Event interface

The Linux network stack provides a standard set of wireless event API for the wireless driver. The WLAN driver uses this API to send the wireless events to application. These wireless events are implemented through a routing Netlink (rtnetlink) socket interface. Any application that wants to listen for these wireless events opens a Netlink socket of type NETLINK\_ROUTE. The hostap driver interface also opens this socket interface for wireless events from the WLAN driver.

### 5.4.1.4 Data Socket interface

Apart from the above two interfaces, the hostap application uses a normal socket interface with the protocol type set to EAPOL (0x888E) to send and receive layer2 EAPOL packets to and from the WLAN driver. The EAPOL packets sent from hostap though this socket gets transmitted through the hard\_start\_xmit transmit interface registered by the driver for WLAN network interface. The EAPOL packet received by the driver is delivered to the network stack through the netif\_rx interface. The network stack appropriately sends this packet to the hostap data socket interface.

### 5.4.1.5 Driver Security Services

The WLAN driver provides the following security-related services:

**Key management support** – provides for primitive encryption methods like WEP, TKIP, and AES (CCMP 128 /256 and GCMP 128/256). The hardware supports a fixed number of key cache entries (mostly 128 entries) to support hardware encryption/decryption. The driver appropriately programs these entries through key management services which includes

- allocating a new key for each node and assigning a corresponding key cache entry in the hardware.
- setting the key value from the hostap in the allocated key entry (in the node structure) and in the assigned key cache entry in the hardware.
- deleting the key value from the node structure and removing the corresponding key cache entry.

**Crypto Encapsulation** – adds appropriate privacy header based on the encryption type for the frames which has the Privacy bit set.

**Crypto Decapsulation** – validates and strips the privacy headers (and trailer) for a received frame that has the privacy bit set. The OS interface layer is notified of the replay failure and MIC failure (for TKIP only) to.

**WPA/RSN capabilities** – maintains RSN/WPA capabilities from the hostap for each VAP, which are used to generate RSN/WPA IEs for the required management frames.

**Software Crypto** – provides support for software encryption and decryption in some error handling cases. This service is also used when the number of stations connected to the AP exceeds the number of key cache entries provided by the hardware. Software Crypto is available only for WEP, TKIP and CCMP 128 encryption methods.

## 5.4.2 Driver Interface for Hostap Implementation

### 5.4.2.1 IOCTL Interface

The WLAN driver registers “ieee80211\_ioctl” function as the ioctl interface for the WLAN network interface (corresponding to each VAP created in the driver). All the ioctls commands for this network interface are handled by this function.

The hostap application uses a set of ioctl commands to interface with the driver. These ioctls commands may be grouped into the following:

- Common ioctl
- AP ioctl commands for hostapd
- Client ioctl commands for wpa\_supplicant
- Wireless event interface
- Driver security services

### 5.4.2.2 Common ioctl commands

The following describes the set of ioctl commands that are common between AP (hostapd) and Client (wpa\_supplicant)

- **Command:** IEEE80211\_IOCTL\_SETKEY

**Function:** ieee80211\_ioctl\_setkey

**Description:** Sets the unicast/multicast keys in the driver through the wlan\_set\_key UMAC API. The key information for setting the key is sent through the “ieee80211req\_key” structure. This structure specifies the key/cipher type and whether the key is to be used for sending and/or receiving. The key index should be set only when working with global keys (use IEEE80211\_KEYIX\_NONE for “no index”).

Otherwise, a unicast/pairwise key is specified by the BSSID (on a station) or MAC address (on an AP). The key length must include any MIC key data; otherwise it should be no more than IEEE80211\_KEYBUF\_SIZE.

- **Command:** IEEE80211\_IOCTL\_DELKEY

**Function:** ieee80211\_ioctl\_delkey

**Description:** Deletes the key in the driver through the wlan\_del\_key UMAC API. The information to delete a key is sent through the structure “ieee80211req\_del\_key”. This structure specifies the key index and MAC address associated with the key that needs to be deleted. Set the key index to IEEE80211\_KEYIX\_NONE when deleting a unicast key.

- **Command:** IEEE80211\_IOCTL\_SET\_APPIEBUF

**Function:** ieee80211\_ioctl\_setappiebuf

**Description:** Function to set the information elements that applications request the driver to add for the specified frame type. Implemented using “wlan\_mlme\_set\_appie” UMAC API. Uses “ieee80211req\_getset\_appiebuf” structure to specify frame type, application buffer length and application buffer to be added.

- **Command:** IEEE80211\_IOCTL\_SETPARAM

**Function:** ieee80211\_ioctl\_setparam

**Description:** Sets various parameters associated with VAP network interface. This command further divided in to sub ioctl commands. Implemented using wlan\_set\_param UMAC API.

- **Subcommand:** IEEE80211\_PARAM\_WPA

**Description:** Set WPA mode i.e WPA or WPA2 or Mixed WPA mode.

- **Subcommand:** IEEE80211\_PARAM\_PRIVACY

**Description:** Enables privacy flag of the VAP to indicate security is enabled on this VAP.

- **Subcommand:** IEEE80211\_PARAM\_AUTHMODE

**Description:** Set the authentication mode to be OPEN or SHARED or 802.1x

- **Subcommand:** IEEE80211\_PARAM\_COUNTERMEASURES

**Description:** Enable TKIP counter measures for authentication rejection.

#### 5.4.2.3 AP ioctl commands for hostapd

- **Command:** IEEE80211\_IOCTL\_SETMLME

**Function:** ieee80211\_ioctl\_setmlme

**Description:** This command provides an interface for the application to perform direct access to MLME layer and performs several MLME actions like sending associate/re associate request, associate/re associate response, authentication, de-authentication, and disassociate management frames. The UMAC provides various APIs to implement these MLME commands. This command will pass the ieee80211req\_mlme structure, which specifies the MLME operation that needs to be performed.

- **Command:** IEEE80211\_IOCTL\_GETKEY

**Function:** ieee80211\_ioctl\_getkey

**Description:** Gets the key from driver using UMAC API wlan\_get\_key. Uses the key index and MAC address associated with that key to get the key value.

- **Command:** IEEE80211\_IOCTL\_FILTERFRAME

**Function:** ieee80211\_ioctl\_setfilter

**Description:** Sets the filter in the OS interface layer to the type frame that the application is interested in receiving from the driver.

- **Command:** IEEE80211\_IOCTL\_GETWPAIE

**Function:** iee80211\_ioctl\_getwpaie

**Description:** Gets the WPA/RSN IE and WPS IE information from the driver. Uses wlan\_node\_getwpaie and wlan\_node\_getwpsie UMAC APIs.

- **Command:** IEEE80211\_IOCTL\_SEND\_MGMT

**Function:** ieee80211\_ioctl\_sendmgmt

**Description:** Sends the management frames directly from the application. Uses wlan\_send\_mgmt UMAC API.

- **Command:** IEEE80211\_IOCTL\_SETPARAM

The following are sub\_ioctl commands that are used by the hostapd applications

- **Subcommand:** IEEE80211\_PARAM\_MCASTCIPHER  
**Description:** Function to set mcast cipher using wlan\_set\_mcast\_ciphers UMAC API.
- **Subcommand:** IEEE80211\_PARAM\_MCASTKEYLEN  
**Description:** Sets mcast key length using wlan\_set\_rsn\_cipher\_param UMAC API.
- **Subcommand:** IEEE80211\_PARAM\_UCASTCIPHERS  
**Description:** Sets unicast cipher using wlan\_set\_unicast\_ciphers UMAC API.
- **Subcommand:** IEEE80211\_PARAM\_KEYMGTALGS  
**Description:** Interface to set key management algorithms for this VAP using wlan\_set\_rsn\_cipher\_param UMAC API.
- **Subcommand:** IEEE80211\_PARAM\_RSNCAPS  
**Description:** Interface to set RSN capabilities such as PREAUTH, MFP required, and so on, to this VAP using wlan\_set\_rsn\_cipher\_param UMAC API.

#### 5.4.2.4 Client ioctl commands for wpa\_supplicant

- **Command:** IEEE80211\_IOCTL\_SCAN\_RESULTS  
**Function:** ieee80211\_ioctl\_getscanresults  
**Description:** Gets the scan results from driver. Uses wlan\_scan\_table\_iterate UMAC API.
- **Command:** IEEE80211\_IOCTL\_SETOPTIE  
**Function:** ieee80211\_ioctl\_setoptie  
**Description:** Sets additional optional information elements for the VAP. Unlike the APPIE interface, the optional IEs set through this interface apply to the VAP, and not any particular frame type. Uses wlan\_mlme\_set\_optie UMAC API.
- **Command:** IEEE80211\_IOCTL\_GETCHANINFO  
**Function:** ieee80211\_ioctl\_getchaninfo  
**Description:** Interface to get the current channel information is set for this VAP. Uses wlan\_get\_chaninfo UMAC API.
- **Command:** IEEE80211\_IOCTL\_GETCHANLIST  
**Function:** ieee80211\_ioctl\_getchanlist  
**Description:** Interface to get current channel list set for this VAP. Uses wlan\_get\_chanlist UMAC API.
- **Command:** IEEE80211\_IOCTL\_GETPARAM  
**Function:** ieee80211\_ioctl\_getparam  
**Description:** Interface to get various parameters associated with the VAP network interface. This command is further divided into sub ioctl commands. Implemented using wlan\_get\_param UMAC API.
  - **Subcommand:** IEEE80211\_PARAM\_COUNTRYCODE

**Description:** Interface to get country code set for this device. Uses wlan\_get\_device\_param UMAC API.

- **Subcommand:** IEEE80211\_IOC\_CONNECTION\_STAT

**Description:** Interface to get the current connection state for a STATION VAP

**Command:** IEEE80211\_IOCTL\_SETPARAM

The following are sub\_ioctlls that are used by the wpa\_supplicant applications

- **Subcommand:** IEEE80211\_PARAM\_UAPSDINFO

**Description:** Interface to get the current UAPSD power save setting for this VAP.

- **Subcommand:** IEEE80211\_PARAM\_DROPUNENCRYPTED

**Description:** Instructs driver to drop all unencrypted DATA packets.

- **Subcommand:** IEEE80211\_PARAM\_CLR\_APPOPT\_IE

**Description:** Interface to clear the application and option IEs that are set by the application through IEEE80211\_IOCTL\_SETAPPIE and IEEE80211\_IOCTL\_SETOPTIE.

- **Subcommand:** IEEE80211\_IOC\_WPS\_MODE

**Description:** Interface to enable the WPS mode in the driver.

- **Subcommand:** IEEE80211\_IOC\_START\_HOSTAP

**Description:** Interface to start the BSS in the driver. Uses wlan\_mlme\_start\_bss UMAC API.

- **Subcommand:** IEEE80211\_PARAM\_NO\_STOP\_DISASSOC

**Description:** Instructs the driver not to send a disassociation frame while stopping the VAP.

- **Subcommand:** IEEE80211\_IOC\_SCAN\_FLUSH

**Description:** Clears the current scan table in the driver. Uses wlan\_scan\_table\_flush UMAC API.

- **Subcommand:** IEEE80211\_PARAM\_PSPOLL

**Description:** Interface for application to send a PSPOLL frame. Uses ieee80211\_send\_pspoll UMAC API.

- **Subcommand:** IEEE80211\_PARAM\_AMPDU

**Description:** Interface to enable AMPDU support in the driver.

- **Subcommand:** IEEE80211\_PARAM\_APBRIDGE

**Description:** Interface to enable the AP bridging support in the driver.

- **Subcommand:** IEEE80211\_PARAM\_MACCMD

**Description:** Interface to set the MAC access control list policy in the driver.

Apart from these IOCTL interfaces, there are a few other IOCTL interfaces used by the wpa\_supplicant that are specific to the P2P features, but those IOCTL interfaces are not described here.

### 5.4.2.5 Wireless Event interface

The following are the various wireless events that the WLAN driver sends for the applications.

- **Event:** IWEVREGISTERED

**Description:** Event to indicate the successful association of a station to the AP.

- **Event:** IWEVCUSTOM

**Description:** This is a custom wireless event and is used to indicate the following events to the applications:

- Replay failure indication
- MIC failure
- PUSH button indication
- Several P2P related events

- **Event:** IWEVASSOCREQIE

**Description:** Event to indicate an association request. But this event is overloaded to send management frame information to the application.

- **Event:** IWEVEXPIRED

**Description:** All events associated with disconnect/deauthentication. Used to indicate

- deauthentication indication for a station or IBSS node.
- disassociation indication for a station or IBSS node.
- station leave indication.

- **Event:** SIOCGIWSCAN

**Description:** Event to indicate scan completion.

- **Event:** SIOCGIWAP

**Description:** Event to indicate connection up or down for STA or IBSS.

- **Event:** IWEVGENIE

**Description:** Used only in P2P mode.

### 5.4.2.6 Driver Security Services

All the security services in the driver are provided through the “crypto” module. This module exports necessary UMAC APIs to support the IOCTLs from the hostap applications. This module is located in drivers/wlan/umac/.

The following is the list of major files under this module and a brief description.

- **File:** ieee80211\_crypto.c

**Description:** Implements attach and detach of crypto module for the VAP and Radio interface. Provides common encapsulate/decapsulation routines for other UMAC modules. Defines UMAC internal functions to set and delete keys.

- **File:** ieee80211\_crypto\_ccmp.c  
**Description:** Implements CCMP and GCMP (AES) specific encapsulation and decapsulation.
- **File:** ieee80211\_crypto\_ccmp\_sw.c  
**Description:** Implements CCMP and GCMP (AES) software encryption and decryption.
- **File:** ieee80211\_crypto\_tkip.c  
**Description:** Implements TKIP specific encapsulation and decapsulation.
- **File:** ieee80211\_crypto\_tkip\_sw.c  
**Description:** Implements TKIP software encryption and decryption.
- **File:** ieee80211\_crypto\_wep.c  
**Description:** Implements WEP specific encapsulation and decapsulation. Also WEP software encryption and decryption.
- **File:** ieee80211\_crypto\_none.c  
**Description:** Dummy crypto functions when CIPHER type is set to none (CLEAR mode).
- **File:** ieee80211\_crypto\_rsn.c  
**Description:** Defines various security-related UMAC APIs for the OS interface layer. Major APIs include
  - wlan\_set\_key and wlan\_del\_key.
  - other UMAC APIs to set and get RSN/WPA capabilities such as unicast/mcast cipher types, auth modes, and so on.

Checking for replay and MIC failure on the received packet is done in the offload layer of the driver.

- **File:** ol\_rx\_pn.c  
**Description:** Does replay check for each packet. If replay is detected, then it drops the packets. This replay check is done based on the security mode of the device.
- **File:** ol\_rx.c  
**Description:** Processes the RX descriptor. If it finds the MIC\_ERROR in the descriptor, then it invokes the ol\_rx\_err() to notify MIC failure to the Hostapd/wpa\_supplicant.

### 5.4.3 Configurations

All the security configurations are accomplished through the hostap configuration files. These files are passed as arguments for the hostapd/wpa\_supplicant application. Refer to the hostap open source website <http://hostap.epitest.fi/> for details about these configuration files.

## 5.5 Enhanced scan results IOCTL to display neighbor AP details

This section describes the functionality to retrieve neighbor AP information. An input/output control (IOCTL) is introduced to obtain the scan result that is required by customers to view neighbor AP details.

The existing IEEE80211\_IOCTL\_SCAN\_RESULTS IOCTL, which retrieves the scan results is enhanced to include a new flag called EXT\_SCAN\_RESULT. To separate the normal wlanconfig ath\* list ap command from this IOCTL that contains the flag to collect extended scan results, the private flag EXT\_SCAN\_RESULT is determined in driver to indicate the IOCTL is to collect neighbor AP details. The specific data structure is appropriately constructed.

The following is the data structure for extended scan results:

```
struct ieee80211req_scan_result_ext
{
    u_int8_t      ssid[32];
    u_int8_t      bssid[6];
    u_int8_t      channel;
    u_int8_t      rssi;
    u_int8_t      capability;
    u_int8_t      tx_chains;
    u_int8_t      rx_chains;
    u_int8_t      dtim_period;
    u_int16_t     beacon_period;
    u_int32_t     max_phryrate;
    u_int32_t     flags;
    u_int32_t     phy_mode;
    u_int32_t     auth_type;
    u_int32_t     kmgmt_type_wpa;
    u_int32_t     encrypt_type_wpa;
    u_int32_t     kmgmt_type_rsn;
    u_int32_t     encrypt_type_rsn;
};
```

The following neighbor AP information is retrieved:

- Flags—Currently, this parameter is used only to indicate if the AP support 11b, 11g. the following is the value:

```
#define SCAN_AP_11B_RATE_SET 0x1
#define SCAN_AP_11G_RATE_SET 0x2
```

- Phy\_mode—Various PHY modes are directly obtained from scan entry. The following is the value:

```
enum ieee80211_phymode {
    IEEE80211_MODE_AUTO          = 0,      /* autoselect */
    IEEE80211_MODE_11A            = 1,      /* 5GHz, OFDM */
    IEEE80211_MODE_11B            = 2,      /* 2GHz, CCK */
    IEEE80211_MODE_11G            = 3,      /* 2GHz, OFDM */
    IEEE80211_MODE_FH             = 4,      /* 2GHz, GFSK */
```

```

        IEEE80211_MODE_TURBO_A          = 5,      /* 5GHz, OFDM, 2x clock
dynamic turbo */

        IEEE80211_MODE_TURBO_G          = 6,      /* 2GHz, OFDM, 2x clock
dynamic turbo */

        IEEE80211_MODE_11NA_HT20        = 7,      /* 5Ghz, HT20 */
        IEEE80211_MODE_11NG_HT20        = 8,      /* 2Ghz, HT20 */
        IEEE80211_MODE_11NA_HT40PLUS    = 9,      /* 5Ghz, HT40 (ext ch +1)
*/
        IEEE80211_MODE_11NA_HT40MINUS   = 10,     /* 5Ghz, HT40 (ext ch -1)
*/
        IEEE80211_MODE_11NG_HT40PLUS    = 11,     /* 2Ghz, HT40 (ext ch +1)
*/
        IEEE80211_MODE_11NG_HT40MINUS   = 12,     /* 2Ghz, HT40 (ext ch -1)
*/
        IEEE80211_MODE_11NG_HT40        = 13,     /* 2Ghz, Auto HT40 */
        IEEE80211_MODE_11NA_HT40        = 14,     /* 5Ghz, Auto HT40 */
        IEEE80211_MODE_11AC_VHT20       = 15,     /* 5Ghz, VHT20 */
        IEEE80211_MODE_11AC_VHT40PLUS   = 16,     /* 5Ghz, VHT40 (Ext ch +1)
*/
        IEEE80211_MODE_11AC_VHT40MINUS  = 17,     /* 5Ghz VHT40 (Ext ch -1)
*/
        IEEE80211_MODE_11AC_VHT40        = 18,     /* 5Ghz, VHT40 */
        IEEE80211_MODE_11AC_VHT80       = 19,     /* 5Ghz, VHT80 */
        IEEE80211_MODE_11AC_VHT160      = 20,     /* 5Ghz, VHT160 */
        IEEE80211_MODE_11AC_VHT80_80    = 21,     /* 5Ghz, VHT80_80 */
};


```

- Max\_phyrate—This parameter indicates the max TX PHY rate that the neighbor AP supports.
- Auth\_type—One of the following values: Open, WEP, WPA, WPA2
- kmgmt\_type\_wpa—One of the following values: PSK, 8021X, both
- encrypt\_type\_wpa—One of the following values: TKIP, AES, both

When the IOCTL is received from a customer that requires neighbor AP information, the code moves to another path to calculate the space that is needed as follows:

```

if ( iwr->u.data.flags & EXT_SCAN_RESULT) {
    req.vap = vap;
    wlan_scan_table_iterate(vap, get_scan_result_ext, (void *)
&req);
} else {
    wlan_scan_table_iterate(vap, get_scan_result, (void *) &req);
}

```

## 5.6 Multiple BSSID

Multiple BSSID or mBSSID is an AP feature that enables creation of multiple virtual access points using a single physical radio. This mechanism enables the configuration of multiple infrastructure networks, each with a different BSSID, and different access characteristics, such as security mode and access control configurations. This effective configuration of round-trip time on APs using UCI commands enables a single physical device to handle multiple VLANs by mapping each “virtual” AP to a VLAN.

A virtual AP (VAP) represents one of the multiple logical APs that run on same physical radio device.

### 5.6.1 Implementation

One of the basic requirements to implement MBSSID is to be able to create multiple MAC addresses (BSSIDs) based on the physical address assigned to the device and expose VAPs as individual network interfaces with individual MAC addresses to the operating system and to the client stations.

The MBSSID feature is based on the ability of the hardware MAC to receive packets with BSSID set to one of a group of addresses that differ only at a preconfigured set of bit positions. This is achieved by setting a mask in the hardware MAC indicating bit positions that it should ignore while filtering the receive packets based on BSSID. By default, 8 virtual APs in an AP are supported with MBSSID. The address of the first VAP is the same as the real MAC address assigned to the device (programmed in EEPROM) and the addresses for subsequent VAPs are generated by setting the ‘Locally Administered’ bit in the MAC address (bit 1 of the first byte) and varying bits 4, 5, and 6 as shown in [Figure 5-5](#).

|                     |                                                       |
|---------------------|-------------------------------------------------------|
| Hardware BSSID Mask | 10001101 11111111 11111111 11111111 11111111 11111111 |
| VAP0 Address        | 00:03:7F:00:01:00                                     |
| VAP1 Address        | 02:03:7F:00:01:00                                     |
| VAP2 Address        | 12:03:7F:00:01:00                                     |
|                     | -                                                     |
|                     | -                                                     |
|                     | -                                                     |
| VAP7 Address        | 52:03:7F:00:01:00                                     |

**Figure 5-5 Multiple BSSID Addresses**

### 5.6.1.1 Direct attach implementation

The BSSID mask shown in [Figure 5-5](#) is programmed in the hardware MAC by using the HAL interface `ath_hal_setbssidmask`. This is done during device initialization in the function `ath_dev_attach` (`lmac/ath_dev/ath_main.c`).

### 5.6.1.2 802.11ac offload implementation

The BSSID for each VAP is sent to the target firmware through the WMI command (`wmi_unified_vdev_create_send` function) as part of the VAP create procedure. Based on all the BSSIDs created, the target firmware generates the appropriate BSSID mask, and initializes the hardware MAC with the appropriate BSSID mask when the VAPs are brought up.

## 5.6.2 VAP management

The basic interface for creating a VAP (`wlan_vap_create`) is same as that of regular AP mode. A new virtual AP interface is created whenever this API is invoked and a BSSID is allocated as explained above. Other BSS management is also same as that of a standard AP mode operation. Refer to the *WLAN AP Modes* chapter for more information on VAP/BSS management and API.

### 5.6.3 Beacon generation

MBSSID support requires generation of beacons for multiple VAPs. Since beacon generation is closely tied with the time synchronization of the BSS, beacons must have generated for all the VAPs at TBTT of the BSS. Since Qualcomm Technologies chips have a single set of beacon timers, this can be achieved by either sending multiple beacons corresponding to multiple BSSs at a common TBTT (*bursting beacons*) or by sending them at alternate time slots by programming hardware beacon timers to indicate TBTT for every VAP individually (*staggered beacons*). The staggered beacons mechanism is preferred as it sends only one beacon for every TBTT indication from hardware and hence timing of beacons are closer to TBTT. But since there is only set of hardware beacon timers available, additional support is required from hardware to set the adjusted timestamps for multiple VAPs, which are required to be closer to multiples of the beacon interval. All our current chips support “TSF Adjust” functionality, which sets the timestamp in beacon frames by adding the internal hardware timer value with the offset that is programmed in the beacon frame by software before queuing the frame for transmission. This functionality is enabled in hardware using the HAL interface `ath_hal_setsfadjust`. Since Qualcomm Technologies chips support this functionality, staggered beacons are used as the default mechanism, though software support for bursted beacons is also available for completeness.

`lmac/ath_dev/ath_beacon.c` contains the implementation of the beacon configuration and beacon generation tasks. The function `ath_beacon_config` allocates and configures beacon slots for each VAP. For staggered beacons, TSF offset is calculated based on the slot allocated for the VAP and is set in beacon frame by the function `ath_beacon_config_tsf_offset`. The hardware MAC adds this offset to the TSF generated by its timer and updates the actual timestamp transmitted in the beacon and probe response frames. Hardware timers are programmed to receive beacon alert interrupts of each VAP and the interface for which beacon is to be generated is determined based on the current hardware TSF timestamp. The rest of the beacon generation procedure is same as that of standard AP mode.

### 5.6.3.1 Transmission and reception

There is no special handling required for transmission since VAPs are exposed as an independent network interface to the operating system, and the state of each VAP required for transmitting data frames is the same as that of a standard AP.

Since the BSSID address mask is set as described above, the hardware receives frames corresponding to all the VAPs and there is no special handling required in the LMAC layer. In the UMAC, the frame is mapped to the VAP interface by means of a client node to VAP association. This is implemented in the function *ath\_net80211\_input* in the file *if\_ath.c*. The client node is identified from the source address of the frame, and each client node points to the VAP corresponding to the BSS to which it belongs. If there is no node associated with the source address of the received frame, the frame will be delivered to all the VAP interfaces and the frame is processed for that particular VAP only, if the BSSID of the VAP matches the RA in the frame.

### 5.6.4 Configure an AP interface

The following command is used in Linux to create an AP interface:

```
wlanconfig ath create wlandev wifi0 wlanmode ap
```

This command creates a virtual AP interface (*ath0*, *ath1*, ...). The basic configuration required for an AP mode operation can be configured using the following command:

```
iwconfig ath0 essid "<AP_SSID>" mode master freq <channel>
iwconfig ath1 essid "<AP2_SSID>" mode master
```

Radio parameters such as channel frequency are common for all the VAPs. Therefore, these parameters need not be set for each VAP individually. Configuring other WLAN parameters can be done in the same way as standard AP mode using *iwpriv* and *iwconfig* commands, passing the interface name corresponding to the virtual AP (*ath0*, *ath0*, ...).

## 5.7 VLAN Support

### 5.7.1 Overview

A Virtual Local Area Network (VLAN) is a group of hosts that communicate as if they were attached to the same broadcast domain, regardless of their physical location. It allows for end stations to be grouped together, even if they are not located on the same network switch.

The WLAN driver implementation supports port-based VLANs (static VLANs) only. Each VAP can be considered a port. Assignments are created by assigning VLAN to VAPs. As a wireless station associates with a VAP, it automatically assumes the VLAN membership of the VAP. No explicit VLAN configuration is required on the station.

### 5.7.2 Terminology

- 802.1Q: A networking standard for the sharing of a physical Ethernet network link by multiple independent logical networks.

- TCI: Tag Control Identifier
- MAC: Media Access Control
- UMAC: Upper MAC
- QDF: Qualcomm Driver Framework
- VAP: Virtual Access Point
- TID: Traffic ID
- AC: Access Category

### 5.7.3 Theory of Operation

During transmission, the WLAN driver checks to see whether the packet is part of a VLAN or if a 802.1Q header is present in the packet. It then extracts the priority from the TCI field and uses it to determine the priority and Traffic Id of the frame.

When a packet is received on a VLAN configured VAP, the WLAN driver attaches to it a VLAN tag using its own VLAN ID and passes it to the upper layers. Only a single VLAN ID can be configured on a VAP at any given time. Hence, a VAP cannot act as a trunk port.

Configurations are carried out using user space utilities.

### 5.7.4 VLAN implementation

The implementation of VLAN support in the driver can be found within the UMAC and QDF modules.

#### 5.7.4.1 QDF

The support in QDF abstracts the process of OS dependent VLAN configuration from WLAN driver. In Linux, `qdf_net_vlan.c` implements and registers the following callbacks:

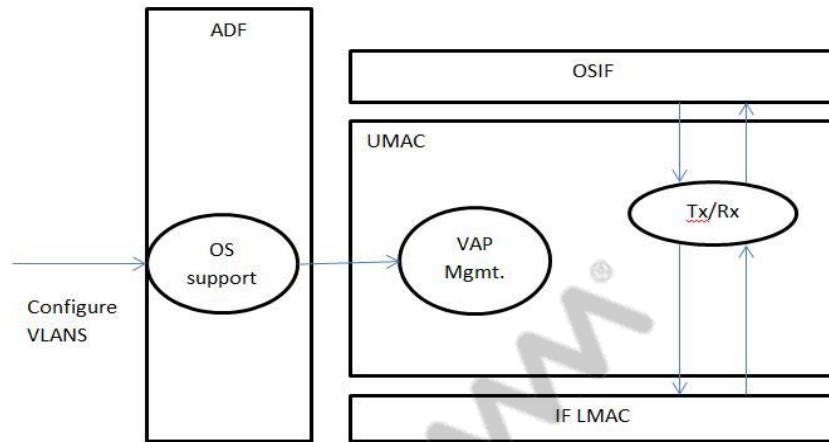
- `ndo_vlan_rx_register`
- `ndo_vlan_rx_add_vid`
- `ndo_vlan_rx_kill_vid`

The following device flags are set during registration:

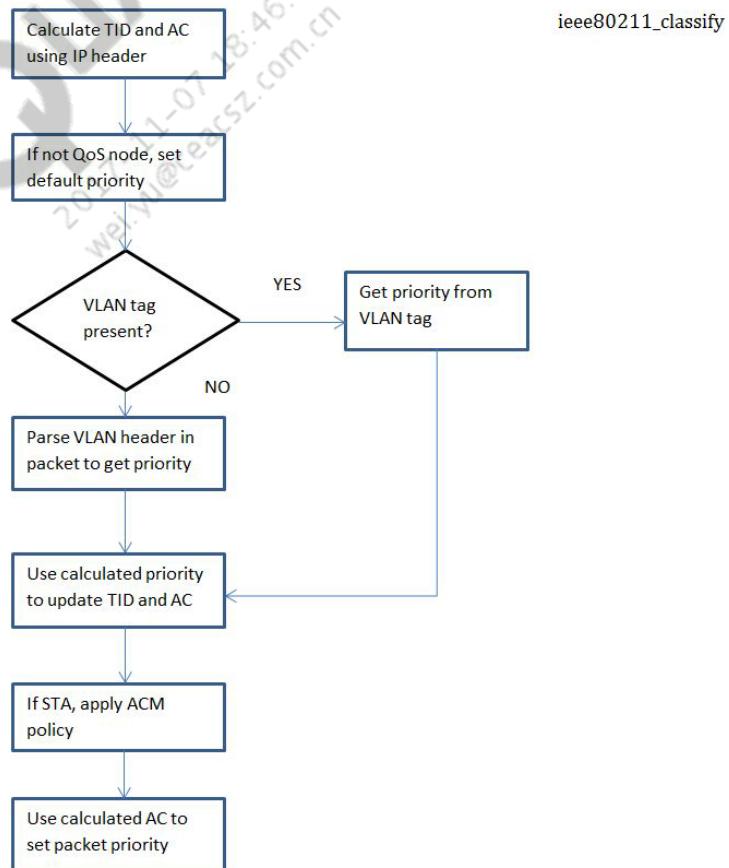
- `NETIF_F_HW_VLAN_TX`
- `NETIF_F_HW_VLAN_RX`
- `NETIF_F_HW_VLAN_FILTER`

#### 5.7.4.2 UMAC

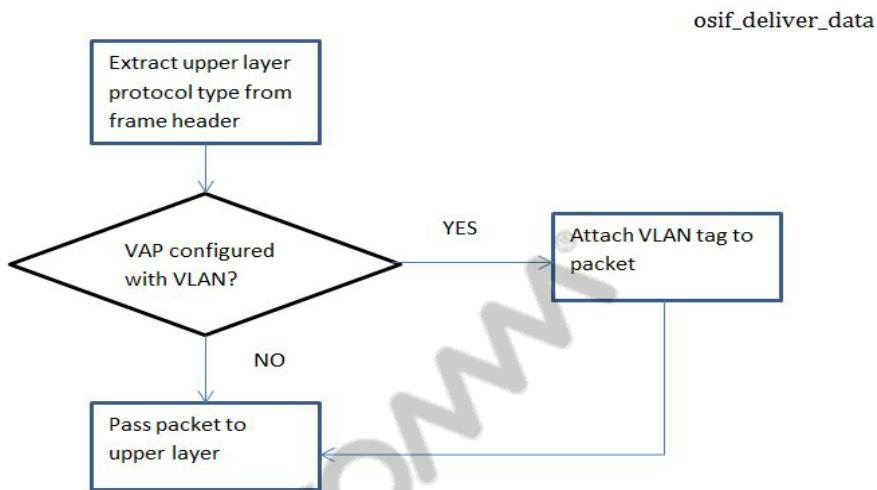
The UMAC and OSIF layers use the configuration on a VAP to determine the priority and traffic ID of outgoing frames and attach VLAN tags to incoming frames. See [Figure 5-6](#).

**Figure 5-6 VLAN Block Diagram**

Egress VLAN processing of packets is done in umac/txrx/Ieee80211\_output.c during execution of the ieee80211\_classify() function. See [Figure 5-7](#).

**Figure 5-7 ieee80211\_classify Function**

Ingress processing is done in osif\_umac.c when osif\_deliver\_data() is executed. See [Figure 5-8](#).



**Figure 5-8 osif\_deliver\_data Function**

#### 5.7.4.3 Full Offload

The Full Offload model of the WLAN software imposes the restriction that VLAN IDs 1 and 2 should not be used for user data as these are used for host-target communication. This restriction applies only when host-target interface is MII.

### 5.7.5 Configuration

Under Linux, configuring VLANs is accomplished by using the ‘vconfig’ utility. It allows users to attach a VLAN to an interface thereby creating a virtual interface. Once a virtual interface is defined, it can be used in the same way as any other interface. For example, these commands define VLANs 2-3 on device ath0-ath1 respectively:

```
# vconfig add ath0 2
# vconfig add ath1 3
```

The following command removes vlan 2 from ath0:

```
# vconfig rem ath0.2
```

The vconfig program can set a variety of other options, including device-naming conventions.

## 5.8 Hy-Fi WLAN

A Hy-Fi device is composed of two elements:

- A device with multiple MAC/PHYs: Wi-Fi, PLC, Ethernet
- A software stack which provides the Hy-Fi functionality

Standard networks assume that there is a single path between any two points. Thus, these networks *do* support devices with multiple interfaces, for example,

- Router with Wi-Fi and Ethernet (and possibly even a PLC dongle)
- Clients which are *either* Wi-Fi or Ethernet or PLC

The clients in these traditional networks use any one of the interfaces – thus guaranteeing a single path between any two points. However, if this single path fails (for example, because of interference on Wi-Fi or a range which is too large) then network connectivity fails. Additionally, these networks are limited in the throughput they provide to the throughput available on that single link (Wi-Fi or PLC).

Hy-Fi networks make use of multiple interfaces concurrently – they are composed of devices with multiple interfaces at both ends of the link, for example:

- Hybrid Router (HR) with Wi-Fi AP and PLC and Ethernet
- Hybrid Clients (HC) with Wi-Fi STA and PLC and Ethernet
  - HC may also have a Wi-Fi AP: in that case it could be called Hybrid Range Extender (HRE) – though effectively it is HC+AP
- Wi-Fi Range Extenders (RE)
- Legacy clients which are Wi-Fi only or PLC only

Hy-Fi networks then support connectivity between HR and HC/HRE on both Wi-Fi and PLC concurrently – using both interfaces to:

- Provide more throughput
- Provide more robust throughput: if there is degradation of one link then traffic can be transmitted on the other link (redundancy).

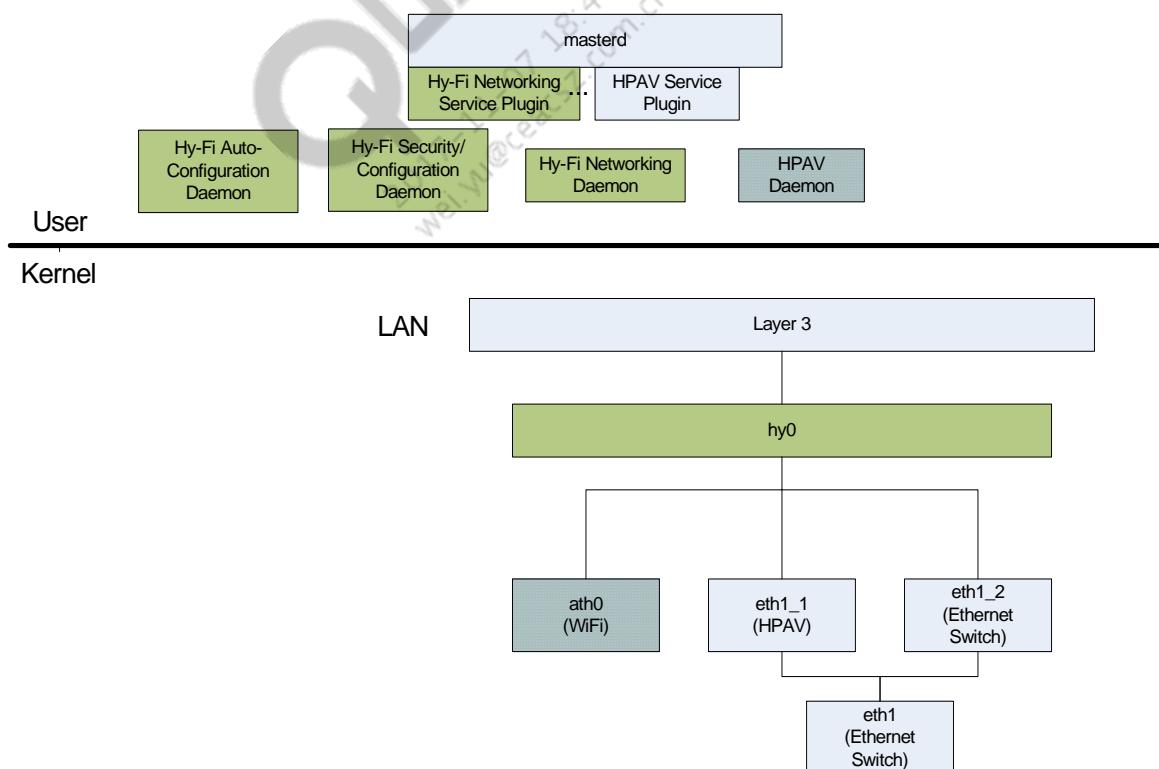
Hy-Fi networking functionality is composed of:

- Infrastructure
  - Topology Discovery: Hy-Fi devices discover each other (including the capabilities and available interfaces of other devices) and the connectivity available to other devices – that is, what links can be used to transmit data to the other device
  - Path Characterization: Hy-Fi devices characterize the Wi-Fi and PLC paths to every other device.
    - This characterization is used to determine how much throughput is available between devices as well as when the medium (Wi-Fi or PLC) is too congested
    - Path characterization is updated at regular intervals (possibly every 1 second) to handle rapidly changing channel conditions and to react rapidly to interference
- Path Selection
  - Stream Identification: Hy-Fi devices identify streams by their Layer 2, 3, and 4 (for example, MAC, IP, TCP) information. Streams are the unit on which the path selection decisions are made.
  - Path Selection: Each stream is assigned to an interface (Wi-Fi or PLC) based on its type (UDP or non-UDP) and the available bandwidth on the Wi-Fi or PLC medium.

- Load Balancing: If a medium (Wi-Fi or PLC) becomes congested, for example, because a Microwave was turned on and caused interference, some or all streams on that medium are moved to other available mediums to relieve the congestion.
- Fail Over: If a medium (Wi-Fi or PLC) completely fails then all streams are moved to other available interfaces.

The Hy-Fi software architecture is shown in below figure. It primarily consists of the following components:

- Kernel space:
  - Hy-Fi Bridge: *hy0*
  - Modifications to drivers
- User space:
  - Hy-Fi Daemon: *hyd*
  - Hy-Fi Security & Configuration Daemon:
  - *wsplcd*
  - Hy-Fi Auto-Configuration Daemon:
  - *acd*



**Figure 5-9 Hy-Fi Software Architecture**

The user space applications and the Hy-Fi bridge *hy0* together provides the Hy-Fi functionality. The Wi-Fi driver basically provides interfaces (ioctls) to the *hy0* bridge and user apps, through

which the driver can be queried to estimate its link capacity or configured to provide alternate paths to reach a destination if one path fails and so on.

The Wi-Fi driver implements four sub features for Hy-Fi functionality

1. Link Metrics collects Hy-Fi specific stats that per destination per access category specific. These stats are collected by Hy-Fi layer at regular intervals to determine the link capacity to that specific node.
2. WMM DSCP Prioritization helps to push packets with specific dscp value to be pushed on to a specific TID.
3. Host Managed Multicast enables Hy-Fi layer to program and manage multicast snooping table entries. Legacy snooping functionality is disabled when this feature gets enabled.
4. Host Managed WDS provides ioctl to Hy-Fi layer through which Hy-Fi layer can manage WDS entries. WDS entries added by Hy-Fi layer called host managed entries, which should not be aged out.

**NOTE** Open a Telnet session to the APUT on port no.7787 for the Band Steering daemon (lbd) debug logs. Open a Telnet session to the APUT on port no.7777 for the -Hy-Fi daemon (hyd) debug logs.

### 5.8.1 API

The Hy-Fi functionality in the driver is enabled or disabled through build flags, ATH\_SUPPORT\_HYFI\_ENHANCEMENTS and ATH\_SUPPORT\_DSCP\_OVERRIDE.

- **ieee80211\_ioctl\_hmmc()** and **ald\_nl\_receive()**: is invoked by the user app to program host managed multicast table entries.
- **ieee80211\_ioctl\_hmwds()** is called by the user app to program host managed WDS entries.
- **ieee80211\_dscp\_override()** is called to identify a TID for a specific packet based on its dscp value.
- **ald\_nl\_receive()** receives netlink messages from the user apps and provides statistics back to user space, also receives host managed multicast tables and programs the driver with the same.
- **ald\_assoc\_notify()** notifies the user apps through netlink messages whenever a station gets associated.
- **ald\_buffull\_notify()** notifies the user app/Hy-Fi layer when driver runs out of descriptor.
- **wlan\_hmwds\_add\_addr()** adds a host managed WDS entry or marks an entry as host managed WDS entry if it already exists. A host managed WDS entry will not age out.
- **wlan\_hmwds\_reset\_addr()** clears the host managed entry flags and entries can age out as normal.

For the offload driver, most of the functionalities are split between the host and the firmware. Link statistics collection happens at the firmware and advertised to host through WMI events. For HMMC, Host programs manage multicast table through WMI commands and firmware performs the multicast to unicast conversion based on the table entries.

For WMM Dscp prioritization, Host programs dscp-tid mapping table in firmware through WMI commands. Firmware identifies packets based dscp values and pushes them onto specific tids.

For HMWDS, Host programs host managed WDS table entries in firmware through WMI commands. Firmware then forwards packets based the host managed WDS table.

- `ol_ath_update_stats_event_handler()`
- `ol_ath_node_collect_stats()`
- `ol_ath_node_reset_ald_stats()`
- `ol_ath_node_collect_stats()` is called to collect statistics for a specific node.
- `ol_ath_node_reset_ald_stats()` resets stats for a specific node.

## 5.8.2 Configuration

The iwpriv options described below are invoked directly by the Hy-Fi user apps and Hy-Fi bridge `hy0` to query and configure Wi-Fi driver.

- **`iwpriv wifix aldstats 1/0`**  
To enable/disable link metrics stats, this should be enabled before testing link metrics feature. This command is applicable only for direct attach VAPs.
- **`wlanconfig ath1 alld sta-enable <sta_mac_addr> <0/1`**  
To enable collection of link metrics stats for the specified STA alone, this command is applicable for offload VAPs alone. This should be enabled before testing link metrics.
- **`wlanconfig ath0 hmwds add_addr <wds_mac_addr> <peer_mac_addr>`**  
To add a managed WDS address through an associated peer.
- **`wlanconfig ath0 hmwds reset_addr <mac_addr>`**  
Resets the given managed WDS address, if `peer_mac_addr` is specified.,  
Resets all the managed WDS entries behind a peer if `<wds_mac_addr>` alone is specified.
- **`wlanconfig ath0 hmwds reset_table`**  
Resets all the managed WDS entries in the global WDS table, if both `<wds_mac_addr>` and `<peer_mac_addr>` are not specified.
- **`wlanconfig ath0 hmwds read_addr <peer_mac_addr>`**  
Lists all the managed WDS addresses behind the given peer
- **`wlanconfig ath0 hmwds read_table`**  
Lists all the managed WDS addresses configured.
- **`iwpriv wifix set_dscp_ovride 1/0`**  
To enable/disable dscp override feature in the ap.
- **`iwpriv wifix get_dscp_ovride`**  
To check whether dscp override is enabled currently
- **`iwpriv wifi0 reset_dscp_map <tid>`**

- To reinitialize all the dscp's with a default tid value. This command is not available for offload vap
- **iwpriv wifi get\_dscp\_tid\_map <dscp>**  
To print the tid configured for a specific dscp value.
  - **iwpriv wifi set\_dscp\_tid\_map <dscp> <tid>**  
To configure a specific tid for specific dscp value.
  - **iwpriv wifi gIgmpDscpOvrid**  
To check whether igmp dscp override is enabled or not
  - **iwpriv athx nopbn 1/0**  
To disable VAPs being notified when jumpstart button gets pushed
  - **iwpriv athx get\_nopbn**  
To check whether nopbn option is enabled for a specific vap
  - **iwpriv wifi sIgmpDscpOvrid 1**  
To enable igmp dscp override.
  - **iwpriv wifi gIgmpDscpTidMap**  
To print current tid configured for igmp packets.
  - **iwpriv wifi sIgmpDscpTidMap <tid>**  
To configure a specific tid for igmp packets

Hy-Fi apps queries/receives Wi-Fi statistics from the driver through a netlink socket are identified by NETLINK\_ALD with IEEE80211\_ALD\_ALL messages. Hy-Fi apps programs managed multicast tables through a netlink message called IEEE80211\_ALD\_MCTBL\_UPDATE

## 5.9 Location wireless interface (LOWI)

The location wireless interface (LOWI) is a comprehensive and unified API used by various technologies that require WiFi measurements and WiFi nodes information. It acts as a central place to handle measurement requests; means that these technologies do not have to interface with WLAN driver directly.

LOWI for QCA\_Networking\_2016.SPF.3.0-based access point (APs) with Qualcomm® Wi-Fi technology has these components:

### LOWI client side library

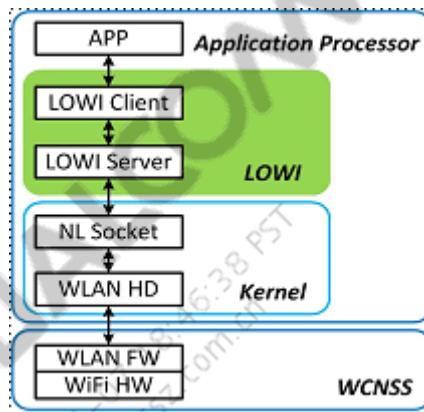
This API is used by clients to send requests to and receive responses from LOWI. The library uses a socket based IPC mechanism to communicate with the LOWI process.

Clients sending requests through the API will receive the status of the request as soon as the request is successfully sent to the IPC hub or fails. Clients receive the response to the requests in a callback after the LOWI process executes the request and generates response and sends it back through the IPC Hub.

## LOWI Server process

The LOWI server process receives requests from clients and performs the requested operation and maintains the resulting scan measurements in a cache. It hosts a LOWIController thread. The LOWIController thread is responsible for receiving requests, processing them and dispatching the responses. It is also responsible for performing cache operations. Additionally, it uses the WifiDriverInterface to communicate with the WifiDriver, provides a layer of abstraction from the actual WifiDriver and has a WLANMeasurementHandler thread that receives the WLAN scan measurements from the WifiDriver.

LOWI process also uses a socket based IPC mechanism to receive requests and send responses to clients through the IPC Hub.



**Figure 5-10** LOWI overview

This document is for programmers who will test LOWI on APs via the command line interface (CLI).

### 5.9.1 LOWI client side library

Figure 5-11 shows the LOWI client side library architecture.

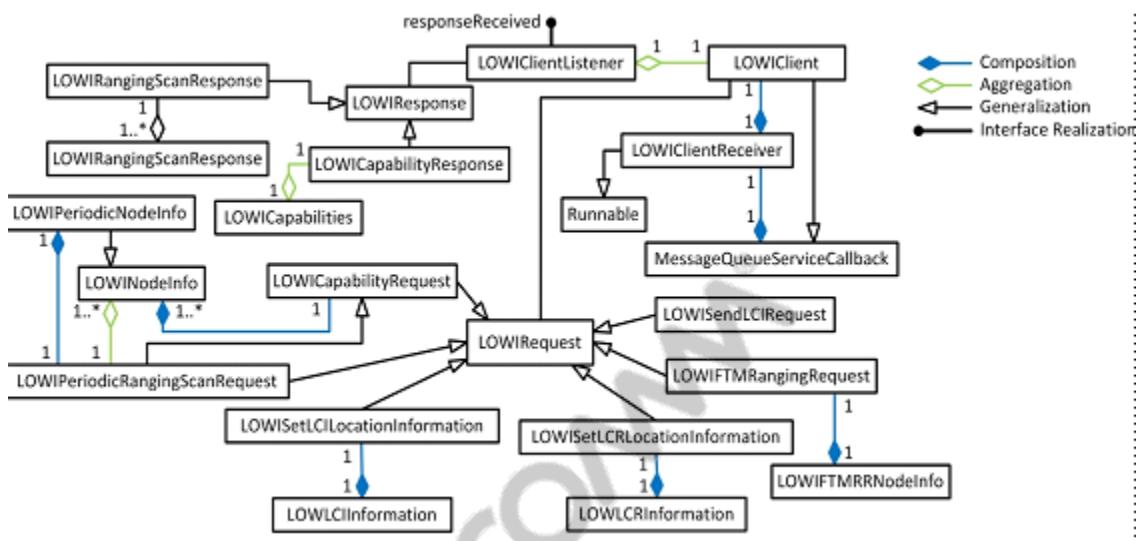
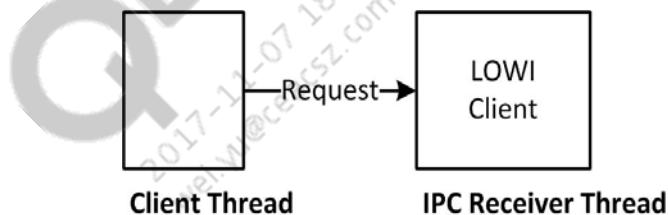


Figure 5-11 LOWI client side library architecture

| Class              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LOWIClient         | <ul style="list-style-type: none"> <li>Exposes the LOWI API clients use to send requests and receive responses from the LOWI server</li> <li>Provides a thin layer to convert the requests and responses to a structured format</li> <li>Uses the <b>LOWIClientReceiver</b> class to help with receiving responses</li> <li>During initialization, creates a separate thread and starts <b>LOWIClientReceiver</b> in that thread</li> <li><b>LOWIClient</b> registers itself as a listener to messages received by the <b>LOWIClientReceiver</b> thread</li> </ul> |
| LOWIClientReceiver | <ul style="list-style-type: none"> <li>Runs in a separate thread and its responsibility is to register with the IPC hub and update the <b>LOWIClient</b> when registration is done</li> <li>Receives messages from the IPC hub and notifies the <b>LOWIClient</b> in its own thread</li> </ul>                                                                                                                                                                                                                                                                     |
| LOWIClientListener | <ul style="list-style-type: none"> <li>Provides a <b>responseReceived</b> interface</li> <li>Called when <b>LOWIClient</b> receives a response for a request from LOWI server</li> <li>Any application that needs to listen to responses from LOWI should implement this interface</li> </ul>                                                                                                                                                                                                                                                                      |

| Class        | Description                                                                            |                                              |
|--------------|----------------------------------------------------------------------------------------|----------------------------------------------|
| LOWIRequest  | A superclass for various requests that will be sent to the LOWI server from the client |                                              |
|              | <b>Supported Data Class</b>                                                            | <b>Takes the client's request for:</b>       |
|              | LOWICapabilityRequest                                                                  | Capability information                       |
|              | LOWIRangingScanRequest                                                                 | Ranging measurements                         |
|              | LOWIPeriodicRangingScanRequest                                                         | Ranging measurements at periodic intervals   |
|              | LOWISetLCILocationInformation                                                          | Setting LCI location information for this AP |
|              | LOWISetLCRLocationInformation                                                          | Setting LCR location information for this AP |
|              | LOWIFTMRangingRequest                                                                  | Sending FTM Ranging requests to a STA        |
|              | LOWISendLCIRequest                                                                     | Sending LCI request to a STA                 |
| LOWIResponse | A superclass for responses that the LOWI client receives from the LOWI server          |                                              |
|              | Supported Data Class                                                                   | Takes the client's request for:              |
|              | LOWICapabilityResponse                                                                 | LOWI capabilities request to the LOWI server |
|              | LOWIRangingScanRequest                                                                 | A ranging scan                               |

### 5.9.1.1 LOWI client side library thread composition



| Thread              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Client Thread       | <ul style="list-style-type: none"> <li>■ Application / client thread that initiates a scan request through the LOWIClient class</li> <li>■ All the requests in LOWIClient are generated in the Client Thread context and sent to the IPC hub</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| IPC Receiver Thread | <ul style="list-style-type: none"> <li>■ Responsible for receiving response messages from IPC hub, converting them to a response structure and providing a callback to the client / application</li> <li>■ All responses to the client are provided in this thread context; LOWIClient responses are executed in this thread's context. The thread also performs client registration with the IPC hub.</li> <li>■ To make sure that no request is received before the registration with the IPC hub is done, the client thread creates and blocks on a BlockingQueue until the registration is completed by the receiver thread. The Receiver thread will retry several times to register with the IPC hub before returning failure.</li> </ul> |

## 5.9.2 LOWI server process

Figure 5-12 shows the LOWI server architecture

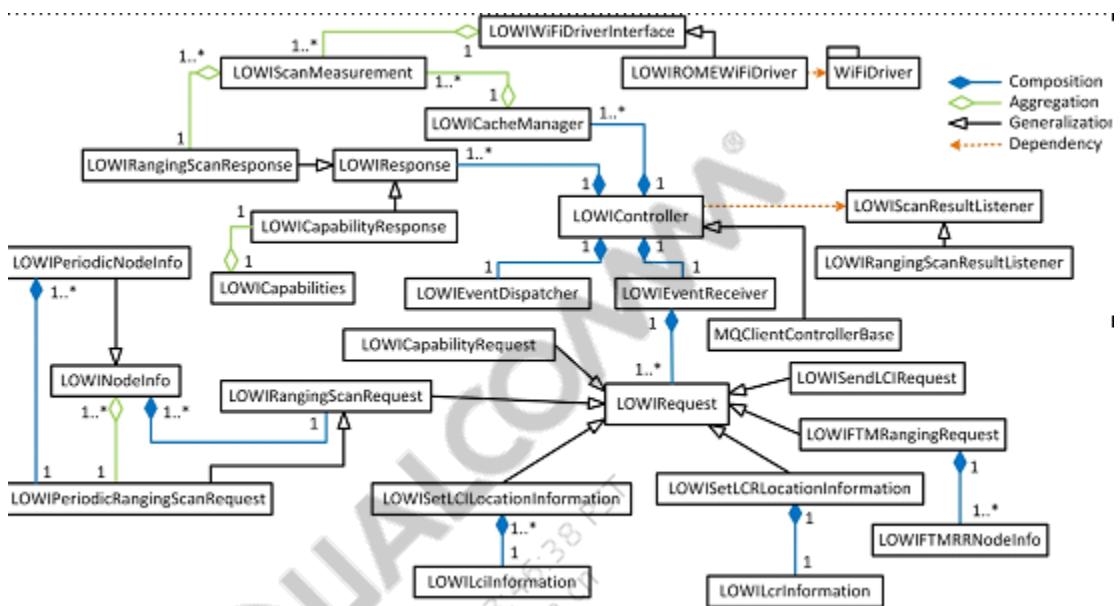


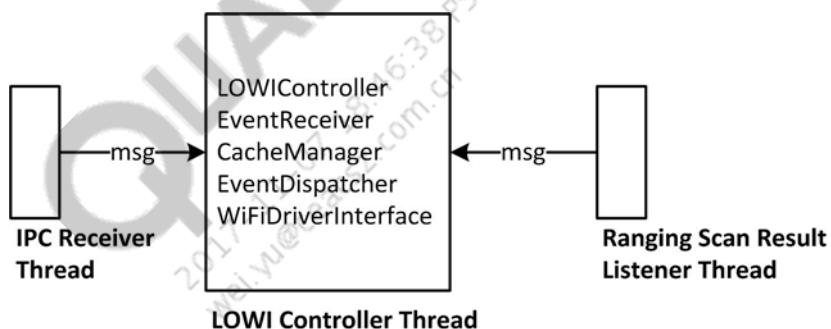
Figure 5-12 LOWI server architecture

| Class           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EventDispatcher | <ul style="list-style-type: none"> <li>Provides functionality to create the IPC messages corresponding to the responses, LOWI supports; it is essentially a shim layer to convert the response to a postcard</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| EventReceiver   | <ul style="list-style-type: none"> <li>Receives the IPC messages from <b>LOWIController</b> class</li> <li>Upon receiving the postcard message it takes appropriate action based on the message type <ul style="list-style-type: none"> <li>If the postcard was a new request it converts the postcard to the requests per request data structures in LOWI. It also extracts the sender identity from the IPC message and puts it into the request data structure.</li> <li>If the postcard contains scan measurements then it invokes the appropriate functionality to start the processing of the measurement results.</li> </ul> </li> <li>It is essentially a shim layer to convert the Postcard into Request / Received Scan Measurement</li> </ul> |

| Class          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LOWIController | <ul style="list-style-type: none"> <li>■ The main controller class in LOWI; responsible for receiving new requests from the IPC hub and measured results from scan results listener threads</li> <li>■ Uses the EventReceiver class to convert the incoming message to either the request / scan measurements</li> <li>■ Maintains a pending queue for all scan requests received from IPC hub; on receiving a new request from the IPC hub it will examine if the request can be serviced right away. If it can, (i.e. from cache) then it will generate the response and send it using EventDispatcher class. <ul style="list-style-type: none"> <li>□ If the request is to trigger a ranging scan, it will be kept in the pending queue until the ongoing scan is completed. After the ongoing scan is completed it will reexamine all the requests in the pending queue and generate responses to all the requests it could service now.</li> <li>□ It will then pick-up the top most pending request and start processing it through the WifiDriverInterface. After which it will block on its IPC message queue to receive a result to the request. Until the result to the request is received in the IPC queue, all the new request will be kept in the pending queue if cannot be served right away.</li> <li>□ It will also trigger a timer for any issued request and if the results are not received and timer expires, then it will no longer wait for the results for the last request and issue another request through WifiDriverInterface. If it does not receive any response to three consecutive requests, then it will close the socket at which scan results listener thread is blocked. This will reset the scan results listener thread to re-establish socket connection.</li> <li>□ On receiving the results from the scan results listener thread, it will try to co-relate the request with the received results on a best match basis. It will then put the results in the cache and generate the response to the current request. It may happen that the results are received but there is no request made i.e. Request was dropped due to time out.</li> <li>□ After the results are cached the requests from the pending queue are served. So it will check all the requests in the pending queue to see if any request can be served right away. Also checks if any request has already expired and drops such requests.</li> </ul> </li> <li>■ This class is the main controller class of LOWI. This class is responsible for receiving the new requests from the IPC Hub and also measurement results from scan results listener threads. In a nutshell: <ul style="list-style-type: none"> <li>□ Creates all major classes of the LOWI (EventReceiver, EventDispatcher, CacheManager, WifiDriverInterface)</li> <li>□ Blocks on IPC Msg Q to receive Request / Measurement</li> <li>□ Extracts the Request and process it.</li> <li>□ Sends Fresh scan request to WifiDriver through WifiDriverInterface.</li> <li>□ Receives measurements and generates responses</li> <li>□ Performs Cache Management</li> <li>□ Reads configurable data from configuration files</li> </ul> </li> </ul> |
| CacheManager   | <ul style="list-style-type: none"> <li>■ CacheManager class is responsible to provide all the cache management functions to the LOWIController.</li> <li>■ CacheManager provides abstraction and we can have any underlying cache management implementation i.e. in memory / sqlite etc</li> <li>■ CacheManager has a CacheHelper class for it's cache operations.</li> <li>■ CacheManager, takes care of all the caching policies in LOWI i.e. One AP has only one measurement in the cache. So it will update if there is already a previous record present for an AP and not create new one. It is the responsibility of the Cache Manager to not allow more than the max number of records (configurable) to be cached</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

| Class                      | Description                                                                                                                                                                                                                                                                                                                                                              |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WifiDriverInterface        | <ul style="list-style-type: none"> <li>Provides the abstraction from the underlying WifiDriver (RIVA/Qualcomm). It is responsible to send the requests to the WiFiDriver.</li> <li>The interface will also expose functionality to its clients to perform RTT Processing and also provide a mechanism to check if batching is required for a ranging request.</li> </ul> |
| ScanResultsListener        | <ul style="list-style-type: none"> <li>Provides the interface with LOWIController to execute the requests and receive results through a separate thread.</li> <li>It provides the framework for the derived classes to implement the request handling and response delivery.</li> </ul>                                                                                  |
| RangingScanResultsListener | <ul style="list-style-type: none"> <li>Responsible to perform the ranging scan and get the results for the scan. This class runs in its own thread.</li> <li>Ranging request is sent through the LOWIWifiDriverInterface to the wifidriver and the results are posted in the post card format to the LOWIController's blocking queue.</li> </ul>                         |
| WiFiDriver                 | <ul style="list-style-type: none"> <li>Actual WiFiDriver API that will be used by the WifiDriverInterface class. This is not the implementation of LOWI</li> </ul>                                                                                                                                                                                                       |

### 5.9.2.1 LOWI server process threads



| Thread              | Description                                                                                                                             |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| IPC Receiver Thread | <ul style="list-style-type: none"> <li>Receives messages from the IPC Hub and send the messages to the IPC controller thread</li> </ul> |

| Thread                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LOWI Controller Thread               | <ul style="list-style-type: none"> <li>■ Receives postcard messages from the IPC receiver thread, Discovery scan result listener thread and Ranging scan result listener thread in its IPC queue. It uses EventReceiver class to convert the post card to Request / Measurement structures. New Request are sent from IPC Receiver thread whereas Discovery / Ranging Scan result Listener threads send the scan measurements.</li> <li>■ This thread is blocked on its IPC msg Queue and retrieves the messages and processes the requests. For any fresh scan it will send the request to the WifiDriver using the WifiDriverInterface to perform a fresh scan. It is the responsibility of this thread to issue only one Request to the WLANController at any time.</li> <li>■ LOWIController spawns a thread for RangingScanResultListener,</li> <li>■ for any fresh ranging scan request, it will send the request to the RangingScanResultsListener thread to get the ranging scan results for the request.</li> <li>■ CacheManager is used in this thread context to perform caching operations.</li> <li>■ EventDispatcher class is used in this thread's context to send responses to the IPC Hub.</li> </ul> |
| Ranging Scan Results Listener Thread | <ul style="list-style-type: none"> <li>■ Remains blocked on its incoming blocking queue waiting for a request. LOWIController sends the ranging scan request to its blocking queue after which this thread gets unblocked and executes the request. This thread issues the ranging scan request to the wifi driver via LOWIWifiDriverInterface and receives the results. It then posts the results to the LOWIController's IPC queue and then blocks on the incoming blocking queue again.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

### 5.9.3 Add a LOWI client on the AP

The LOWI server can only operate when Wi-Fi is available. It is the client's responsibility to wait till at least one VAP is available before sending a request to the LOWI server. Further, clients are responsible for aggregating any data available from LOWI.

#### 5.9.3.1 LOWI client dependencies

LOWI clients are dependent on the qca-lowi package and client library.

Add the following feeds to the Makefile to ensure this:

- To ensure qca-lowi package is built before client package:

```
PKG_BUILD_DEPENDS:=qca-lowi
```

- To ensure qca-lowi packages is built/installed before this package add qca-lowi to DEPENDS entry for package/\*

```
define Package/$(PKG_NAME)
:
DEPENDS:=qca-lowi ...
:
endef
```

#### 5.9.3.2 Makefile sample for Lowi-client

- Headers for theLOWI client library is available at \$(PKG\_BUILD\_DIR)/internal/lowi/inc
- LOWI client library liblowi-client.so is available at /usr/lib/lowi

```
SHARED_LIB_PATH= -L$(STAGING_DIR)/usr/lib/lowi -Wl,-rpath=/usr/lib/lowi
SHARED_LIBS= liblowi-client -lpthread
```

```

INC_DIRS+= -I$(STAGING_DIR)/usr/include/liblowi_client
CFLAGS+= ${INC_DIRS}
%.o: %.cpp
    $(CXX) -c -o $@ $(CFLAGS) ${COPTS} $<
LIB_OBJS=$(patsubst %.cpp,%.o,$(wildcard *.cpp))
lowi-sample-test: $(LIB_OBJS) $(STATIC_LIBS)
    $(CXX) $(SHARED_LIB_PATH) -o $@ $(CFLAGS) $(LIB_OBJS) $(SHARED_LIBS)

```

### 5.9.3.3 Sample source code

```

#include <stdio.h>
#include <unistd.h>
#include <inc/lowi_client.h>
using namespace qc_loc_fw;
class LowiClientListenerImpl : public LOWIClientListener
{
    void responseReceived(LOWIResponse *response)
    {
        printf("response received..\n");
        if (response->getResponseType() == LOWIResponse::RANGING_SCAN)
        {
            LOWIRangingScanResponse *resp = (LOWIRangingScanResponse
*)response;
            vector<LOWIScanMeasurement *> scanMeasurements = resp-
>scanMeasurements;
            // use scan measurements
        }
    }
};

int main(int argc, char *argv[])
{
    LOWIClientListener *listener = new LowiClientListenerImpl();
    LOWIClient *client = LOWIClient::createInstance(listener, true,
LOWIClient::LL_INFO);
    LOWIMacAddress mac(0, 1, 2, 3, 4, 5);
    vector<qc_loc_fw::LOWIPeriodicNodeInfo> scan_params;

    //fill scan_params
    LOWIPeriodicNodeInfo ap_info;
    LOWIMacAddress mac(0x8c, 0xfd, 0xf0, 0x01, 0xe6, 0xbd);
    ap_info.bssid.setMac(mac);
    ap_info.rttType = RTT2_RANGING;
    ap_info.bandwidth = qc_loc_fw::BW_20MHZ;
    ap_info.band_center_freq1 = 5500;
    ap_info.band_center_freq2 = 0;
    LOWIChannelInfo chan = LOWIChannelInfo(100,
LOWIDiscoveryScanRequest::BAND_ALL);
    ap_info.frequency = chan.getFrequency();
    FTM_SET_BURST_DUR(ap_info.ftmRangingParameters, 15);
    scan_params.push_back(ap_info);

    LOWIPeriodicRangingScanRequest *ran = new
LOWIPeriodicRangingScanRequest(1, scan_params, 0);
    if (NULL != ran)
    {

```

```

        if (LOWIClient::STATUS_OK != client->sendRequest(ran))
        {
            printf("fail..\n");
        }
        else
        {
            int i = 6;
            while (i--)
            {
                sleep(10);
            }
        }
        delete ran;
        if (NULL != client)
        {
            delete client;
            client = NULL;
        }

        if (NULL != listener)
        {
            delete listener;
            listener = NULL;
        }
        return 0;
    }
}

```

## 5.9.4 Client side library classes

### 5.9.4.1 qc\_loc\_fw::LOWIClient class reference

Inherits MessageQueueServiceCallback.

#### Detailed Description

API for sending various requests to LOWI for a LOWIClient. It also provides information on LOWI's capabilities.

#### Public Types

- enum eRequestStatus {STATUS\_OK, BAD\_PARAMS, SEND\_FAILURE}
- enum eLogLevel {LL\_LOG\_OFF = 0, LL\_ERROR = 1, LL\_WARNING = 2, LL\_INFO = 3, LL\_DEBUG = 4, LL\_VERBOSE = 5, LL\_LOG\_ALL = 100}

#### Public Member Functions

- virtual ~LOWIClient ()
- eRequestStatus sendRequest (LOWIREquest \*const request)
- LOWICapabilityResponse \* getCapabilities (LOWIREquest \*const request)

- `LOWIResponse * processRequest (LOWIRequest *const request)`

### Static Public Member Functions

- `static LOWIClient * createInstance (LOWIClientListener *const ptr, bool enableLogging=false, eLogLevel log_level=LL_INFO)`
- `static bool setLogLevel (eLogLevel log_level)`

### Member function documentation

#### `enum qc_loc_fw::LOWIClient::eLogLevel`

Defines the log level.

| Enumerator              |               |
|-------------------------|---------------|
| <code>LL_LOG_OFF</code> | No logging    |
| <code>LL_ERROR</code>   | Error level   |
| <code>LL_WARNING</code> | Warning level |
| <code>LL_INFO</code>    | Info level    |
| <code>LL_DEBUG</code>   | Debug level   |
| <code>LL_VERBOSE</code> | Verbose level |
| <code>LL_LOG_ALL</code> | Log all       |

#### `enum qc_loc_fw::LOWIClient::eRequestStatus`

Defines the status of the request.

| Enumerator                |                                   |
|---------------------------|-----------------------------------|
| <code>STATUS_OK</code>    | Request successfully sent to LOWI |
| <code>BAD_PARAMS</code>   | Error due to bad parameters       |
| <code>SEND_FAILURE</code> | Failure to send request to LOWI   |

### Constructor and destructor documentation

#### `virtual qc_loc_fw::LOWIClient::~LOWIClient ()[virtual]`

Destructor

## Member function documentation

**static LOWIClient\* qc\_loc\_fw::LOWIClient::createInstance (LOWIClientListener \*const ptr, bool enableLogging = false, eLogLevel log\_level = LL\_INFO)[static]**

Creates instance of LOWIClient and initializes it. It is a blocking call until the Instance is created and registered with the IPC hub.

| Parameters          | Description                                                                                                                              |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| LOWIClientListener* | Pointer to the LOWIClientListener which will be notified when the response is received                                                   |
| bool                | Enable / disable logging (optional).                                                                                                     |
| eLogLevel           | Log level at which the library should be logging. This will only take effect if the logging is enabled in previous parameter (optional). |

**LOWICapabilityResponse\* qc\_loc\_fw::LOWIClient::getCapabilities (LOWIRequest \*const request)**

Sends request for capabilities created by the lowi client application. Blocks until the capabilities are received or a timeout occurs if the capabilities do not come. The timeout is defined by CAPABILITIES\_REQ\_BLOCKING\_TIMEOUT\_SEC which is set to 5 seconds by default. This will be the maximum time the client will have to wait if no response is generated by LOWIController.

| Parameters | Description             |
|------------|-------------------------|
| request    | request to be processed |

### Returns

LOWICapabilityResponse\*: ptr to capabilities response

**LOWIResponse\* qc\_loc\_fw::LOWIClient::processRequest (LOWIRequest \*const request)**

Process a request created by the client. Checks the request type and blocks on the following request types until a corresponding response is returned. Blocking timeout is defined by STATUS\_BLOCKING\_TIMEOUT\_SEC and is currently set to 5 seconds by default.

| Parameters   | Description             |
|--------------|-------------------------|
| LOWIRequest* | request to be processed |

### Returns

LOWIResponse\*: pointer to LOWIResponse

**eRequestStatus qc\_loc\_fw::LOWIClient::sendRequest (LOWIRequest \*const request)**

Sends the request created by the client.

| Parameters   | Description             |
|--------------|-------------------------|
| LOWIRequest* | Request to be processed |

**Returns**

eReturnStatus Return status of the request

**static bool qc\_loc\_fw::LOWIClient::setLogLevel (eLogLevel log\_level)[static]**

Sets logging level for LOWIClient library.

| Parameters | Description                                       |
|------------|---------------------------------------------------|
| eLogLevel  | Log level at which the library should be logging. |

**Returns**

True for success, false otherwise

### 5.9.4.2 qc\_loc\_fw::LOWIClientListener class reference

#### Detailed Description

This class should be sub classed by the client of LOWI to receive responses to the requests made through LOWIClient.

#### Public Member Functions

- virtual void responseReceived (LOWIResponse \*response)=0
- virtual ~LOWIClientListener ()=0

#### Constructor and destructor documentation

**virtual qc\_loc\_fw::LOWIClientListener::~LOWIClientListener ()[pure virtual]**

Destructor

#### Member function documentation

**virtual void qc\_loc\_fw::LOWIClientListener::responseReceived (LOWIResponse \* response)[pure virtual]**

Notifies the listener whenever the Response is received for a Request sent by the client.

**NOTE** The notification happens in a different thread from the thread in which the request was made by the client.

| Parameters    | Description       |
|---------------|-------------------|
| LOWIResponse* | Response received |

Client upon receiving this callback should inspect the type of response by response->getResponseType () and then type cast the response to retrieve the response data.

i.e. Sample usage of a capabilities response is

```
if (response->getResponseType () == LOWIResponse.CAPABILITY) {LOWICapabilities* cap =
((LOWICapabilityResponse)response)->getCapabilities();}
```

The documentation for this class was generated from the following file:

- lowi\_client.h

### 5.9.4.3 qc\_loc\_fw::LOWIClientReceiver Class Reference

Inherits Runnable.

#### Detailed description

This class receives the messages from the IPC Hub. Provides socket based IPC implementation. This class is run in a separate thread. Provides the response to the Listener.

#### Public member functions

- [LOWIClientReceiver](#)(const char \*const socket\_name, BlockingQueue \*const pLocalMsgQueue, MessageQueueClient \*const conn, MessageQueueServiceCallback \*const callback, const char \*const client\_name)
- virtual [~LOWIClientReceiver](#)()
- bool [init](#)()

#### Static Public Attributes

- static const char \*const [TAG](#)

#### Protected Member Functions

- virtual void [run](#)()

## Constructor and destructor documentation

**qc\_loc\_fw::LOWIClientReceiver::LOWIClientReceiver (const char \*const socket\_name, BlockingQueue \*const pLocalMsgQueue, MessageQueueClient \*const conn, MessageQueueServiceCallback \*const callback, const char \*const client\_name)**

Constructor LOWIClientReceiver

| Parameters                   | Description                                               |
|------------------------------|-----------------------------------------------------------|
| char*                        | Name of the server socket to connect to                   |
| BlockingQueue*               | IPC BlockingQueue to communicate with main thread         |
| MessageQueueClient*          | Pointer to MessageQueueClient to communicate with IPC Hub |
| MessageQueueServiceCallback* | Pointer for callback on receiving the msg from IPC hub    |
| char*                        | Name of this client. Ideal choice is the thread ID.       |

**virtual qc\_loc\_fw::LOWIClientReceiver::~LOWIClientReceiver ()[virtual]**

Destructor

## Member function documentation

**bool qc\_loc\_fw::LOWIClientReceiver::init ()**

Blocking call to initialize the LOWIClientReceiver. Starts the thread internally and does register with IPC hub. Calling thread blocks until the registration is done.

### Returns

True if success, false otherwise.

**virtual void qc\_loc\_fw::LOWIClientReceiver::run ()[protected], [virtual]**

Runs the main loop of the thread. This will internally run the blocking loop of the MessageQueueClient to receive messages from the IPC hub and provide callback in this thread context to the MessageQueueServiceCallback

## Member data documentation

**const char\* const qc\_loc\_fw::LOWIClientReceiver::TAG[static]**

Log Tag

### 5.9.4.4 qc\_loc\_fw::LOWICapabilities class reference

#### Public Member Functions

- LOWICapabilities ()

## Public attributes

- bool discoveryScanSupported
- bool rangingScanSupported
- bool activeScanSupported
- bool oneSidedRangingSupported
- bool dualSidedRangingSupported11v
- bool dualSidedRangingSupported11mc
- uint8 bwSupport
- uint8 preambleSupport
- uint32 supportedCapablities

## Detailed Description

This class defines the Capabilities of the WifiDriver

## Constructor and destructor documentation

### **LOWICapabilities::LOWICapabilities ()**

Constructor

## Member data documentation

### **bool qc\_loc\_fw::LOWICapabilities::activeScanSupported**

True if Active discovery scan is supported. Not supported for Access Point devices

### **uint8 qc\_loc\_fw::LOWICapabilities::bwSupport**

Highest bandwidth support for rtt requests

| Bandwidth | Value |
|-----------|-------|
| 20 MHz    | 0     |
| 40 MHz    | 1     |
| 80 MHz    | 2     |
| 160MHz    | 3     |

### **bool qc\_loc\_fw::LOWICapabilities::discoveryScanSupported**

True if Discovery scan is supported. Not supported for Access Point devices

### **bool qc\_loc\_fw::LOWICapabilities::dualSidedRangingSupported11mc**

True if dual-sided ranging per 802.11mc FTM standard is supported

**bool qc\_loc\_fw::LOWICapabilities::dualSidedRangingSupported11v**

True if dual-sided ranging per 11v std is supported

**bool qc\_loc\_fw::LOWICapabilities::oneSidedRangingSupported**

True if single-sided ranging is supported

**uint8 qc\_loc\_fw::LOWICapabilities::preambleSupport**

Bit mask representing preambles supported for RTT requests

**bool qc\_loc\_fw::LOWICapabilities::rangingScanSupported**

True if ranging scan is supported.

**uint32 qc\_loc\_fw::LOWICapabilities::supportedCapablities**

Bit mask representing capability supported for the loaded driver.

The following bitmasks indicate supported capability:

| Capability               | Bitmask |
|--------------------------|---------|
| No scan supported        | 0x00    |
| LP scan supported        | 0x01    |
| Discovery scan supported | 0x02    |
| Ranging scan Supported   | 0x03    |
| BG scan supported        | 0x04    |

### 5.9.4.5 qc\_loc\_fw::LOWICapabilityRequest class reference

Inherits qc\_loc\_fw::LOWIRequest.

#### Detailed Description

Capability request

#### Public member functions

- LOWICapabilityRequest (uint32 requestId)
- virtual ~LOWICapabilityRequest ()
- virtual eRequestType getRequestType () const

## Constructor and destructor documentation

### **LOWICapabilityRequest::LOWICapabilityRequest (uint32 requestId)**

Constructor

| Parameters | Description                                                                                |
|------------|--------------------------------------------------------------------------------------------|
| uint32     | Request id generated by the client This will be echoed back in the corresponding response. |

### **LOWICapabilityRequest::~LOWICapabilityRequest ()[virtual]**

Destructor

## Member function documentation

### **LOWIRequest::eRequestType LOWICapabilityRequest::getRequestType () const[virtual]**

Returns the request type

#### **Returns**

eRequestType type of request

Implements qc\_loc\_fw::LOWIRequest.

## 5.9.4.6 qc\_loc\_fw::LOWICapabilityResponse class reference

Inherits qc\_loc\_fw::LOWIResponse.

### Detailed Description

Response to the Capability Request

### Public Member Functions

- LOWICapabilityResponse (uint32 requestId, LOWICapabilities capabilities, bool status)
- virtual ~LOWICapabilityResponse ()
- virtual eResponseType getResponseType ()
- LOWICapabilities getCapabilities ()
- bool getStatus ()

## Constructor and destructor documentation

**LOWICapabilityResponse::LOWICapabilityResponse (uint32 requestId,  
LOWICapabilities capabilities, bool status)**

Constructor

| Parameters       | Description                                   |
|------------------|-----------------------------------------------|
| uint32           | requestId generated by the client for request |
| LOWICapabilities | Capabilities object                           |
| bool             | true for success, false to indicate failure   |

**LOWICapabilityResponse::~LOWICapabilityResponse ()[virtual]**

Destructor

## Member function documentation

**LOWICapabilities LOWICapabilityResponse::getCapabilities ()**

Return the capabilities

### Returns

LOWICapabilities

**LOWIResponse::eResponseType LOWICapabilityResponse::getResponseType ()[virtual]**

Returns the response type

### Returns

eResponseType Type of Response

Implements qc\_loc\_fw::LOWIResponse.

**bool LOWICapabilityResponse::getStatus ()**

Returns the status of the request

### Returns

Bool true for success, false otherwise

### 5.9.4.7 qc\_loc\_fw::LOWIFTMRangingRequest class reference

Inherits qc\_loc\_fw::LOWIRequest.

## Detailed Description

FTM Ranging Request used by LOWI Client to request FTM ranging from associated STA

## Public Member Functions

- **LOWIFTMRangingRequest (uint32 requestId, LOWIMacAddress bssid, uint16 randInterval, vector< LOWIFTMRRNodeInfo > &nodes)**
- **virtual ~LOWIFTMRangingRequest ()**
- **const LOWIMacAddress & getBSSID () const**
- **const vector< LOWIFTMRRNodeInfo > & getNodes () const**
- **uint16 getRandInter () const**
- **virtual eRequestType getRequestType () const**

## Constructor and destructor documentation

**LOWIFTMRangingRequest::LOWIFTMRangingRequest (uint32 requestId, LOWIMacAddress bssid, uint16 randInterval, vector< LOWIFTMRRNodeInfo > & nodes)**

Constructor

| Parameters                  | Description                                                                           |
|-----------------------------|---------------------------------------------------------------------------------------|
| uint32                      | Request id generated by the client This is echoed back in the corresponding response. |
| LOWIMacAddress              | BSSID of the STA that is being requested to do the ranging request                    |
| uint16                      | Randomization interval as per defined in 802.11mc                                     |
| vector< LOWIFTMRRNodeInfo > | Vector of nodes containing list of APs to range with                                  |

**LOWIFTMRangingRequest::~LOWIFTMRangingRequest ()[virtual]**

Destructor

## Member function documentation

**const LOWIMacAddress & LOWIFTMRangingRequest::getBSSID () const**

Returns the BSSID of the STA that is being requested to do the ranging request.

## Returns

LOWIMacAddress &

**const vector< LOWIFTMRRNodeInfo > & LOWIFTMRangingRequest::getNodes () const**

Returns the vector of nodes containing list of APs to range with for this request

#### >Returns

vector<LOWIFTMRRNodeInfo> &

**uint16 LOWIFTMRangingRequest::getRandInter () const**

Returns randomization interval for this request

#### >Returns

uint16

**LOWIRequest::eRequestType LOWIFTMRangingRequest::getRequestType () const[virtual]**

Returns the request type

#### >Returns

eRequestType type of request

Implements qc\_loc\_fw::LOWIRequest.

### 5.9.4.8 qc\_loc\_fw::LOWIFTMRRNodeInfo Struct Reference

#### Detailed Description

Contains AP information for each AP with which LOWI client want to range with for FTMRR

#### Public Member Functions

- LOWIFTMRRNodeInfo (LOWIMacAddress bssid, uint32 bssidInfo, uint8 operatingClass, uint8 phyType, uint8 ch, uint8 center\_Ch1, uint8 center\_Ch2, eRangingBandwidth bandwidth)
- LOWIFTMRRNodeInfo (LOWIMacAddress bssid, uint32 bssidInfo, uint8 operatingClass, uint8 phyType, uint8 ch)

#### Public Attributes

- LOWIMacAddress bssid
- uint32 bssidInfo
- uint8 operatingClass
- uint8 phyType
- uint8 ch

- uint8 center\_Ch1
- uint8 center\_Ch2
- eRangingBandwidth bandwidth

### 5.9.4.9 qc\_loc\_fw::LOWILciInformation Struct Reference

#### Detailed Description

LCI Information of an AP

#### Public Member Functions

- LOWILciInformation ()

#### Public Attributes

- int64 latitude
- int64 longitude
- int32 altitude
- uint8 latitude\_unc
- uint8 longitude\_unc
- uint8 altitude\_unc
- eLowiMotionPattern **motion\_pattern**
- int32 **floor**
- int32 height\_above\_floor
- int32 height\_unc

#### Constructor and destructor documentation

##### LOWILciInformation::LOWILciInformation ()

Constructor

### 5.9.4.10 qc\_loc\_fw::LOWILcrInformation Struct Reference

#### Detailed description

LCR Information on an AP

#### Public member functions

- LOWILcrInformation ()

### Public Attributes

- uint8 country\_code [LOWI\_COUNTRY\_CODE\_LEN]
- uint32 length
- INT8 **civic\_info** [CIVIC\_INFO\_LEN]

### Constructor and destructor documentation

**LOWILcrInformation::LOWILcrInformation ()**

Constructor

## 5.9.4.11 qc\_loc\_fw::LOWILocationIE Struct Reference

### Detailed description

Beacon information element used to store LCI/LCR data

### Public member functions

- **LOWILocationIE ()**
- **LOWILocationIE (const LOWILocationIE &rhs)**
- **~LOWILocationIE ()**

### Public attributes

- uint8 **id**
- uint8 **len**
- uint8 \* **locData**

### Constructor and destructor documentation

**LOWILocationIE::LOWILocationIE ()**

Constructor and copy constructor

**LOWILocationIE::~LOWILocationIE ()**

Destructor

## 5.9.4.12 qc\_loc\_fw::LOWIMacAddress Class Reference

### Detailed description

Class for LOWIMacAddress

## Public member functions

- LOWIMacAddress ()
- LOWIMacAddress (const unsigned char \*const pAddr)
- LOWIMacAddress
- (uint8 a0, uint8 a1, uint8 a2, uint8 a3, uint8 a4, uint8 a5)
- LOWIMacAddress (const LOWIMacAddress &rhs)
- unsigned int getLo24 () const
- unsigned int getHi24 () const
- unsigned long long getFull48 () const
- unsigned long long getReversed48 () const
- int setMac (const LOWIMacAddress &rhs)
- int setMac (const unsigned char \*const pAddr)
- int setMac (const int addr\_hi24, const int addr\_lo24)
- int compareTo (const LOWIMacAddress &mac)
- unsigned char operator[] (const int index) const
- void print ()
- LOWIMacAddress & operator= (const LOWIMacAddress &rhs)
- bool isLocallyAdministeredMac ()

## Static public member functions

- static int compareTo (const void \*pLhs, const void \*pRhs)

## Static public attributes

- static const char \*const TAG = "LOWIMacAddress"

## Constructor and destructor documentation

### **LOWIMacAddress::LOWIMacAddress ()**

Default constructor

### **LOWIMacAddress::LOWIMacAddress (const unsigned char \*const pAddr)**

Constructor

| Parameters | Description       |
|------------|-------------------|
| unsigned   | char* Mac address |

**LOWIMacAddress::LOWIMacAddress (uint8 a0, uint8 a1, uint8 a2, uint8 a3, uint8 a4, uint8 a5)**

Constructor

| Parameters | Description                |
|------------|----------------------------|
| uint8      | first byte of the address  |
| uint8      | second byte of the address |
| uint8      | third byte of the address  |
| uint8      | fourth byte of the address |
| uint8      | fifth byte of the address  |
| uint8      | sixth byte of the address  |

**LOWIMacAddress::LOWIMacAddress (const LOWIMacAddress & rhs)**

Copy constructor

| Parameters      | Description |
|-----------------|-------------|
| LOWIMacAddress& | Mac address |

**Member function documentation****int LOWIMacAddress::compareTo (const void \* pLhs, const void \* pRhs)[static]**

Compares Mac addresses

| Parameters | Description    |
|------------|----------------|
| void*      | LOWIMacAddress |
| void*      | LOWIMacAddress |

**Returns**

Comparison results (0 for equal)

**int LOWIMacAddress::compareTo (const LOWIMacAddress & mac)**

Compares Mac addresses

| Parameters      | Description    |
|-----------------|----------------|
| LOWIMacAddress& | LOWIMacAddress |

**Returns**

Comparison results (0 for equal)

**unsigned long long LOWIMacAddress::getFull48 () const**

Returns the Full 48 bits of the Mac Address

**Returns**

Long containing full 48 bits of the Mac address

**`unsigned int LOWIMacAddress::getHi24 () const`**

Returns the Upper 24 bits

**Returns**

Integer containing upper 24 bits of the Mac address

**`unsigned int LOWIMacAddress::getLo24 () const`**

Returns the Lower 24 bits

**Returns**

Integer containing lower 24 bits of the Mac address

**`unsigned long long LOWIMacAddress::getReversed48 () const`**

Returns the full 48 bits of the Mac address in reverse order

**Returns**

Long containing full 48 bits of the Mac address

**`bool LOWIMacAddress::isLocallyAdministeredMac ()`**

Checks if the Mac address is Locally administered or not

**Returns**

False - Globally administered, true - Locally administered

**`LOWIMacAddress & LOWIMacAddress::operator= (const LOWIMacAddress & rhs)`**

Copies the values from passed LOWIMacAddress

| Parameters                       | Description                 |
|----------------------------------|-----------------------------|
| <code>LOWIMacAddress&amp;</code> | <code>LOWIMacAddress</code> |

**Returns**

`LOWIMacAddress&`

**unsigned char LOWIMacAddress::operator[] (const int index) const**

Returns the value at the index passed

| Parameters | Description                   |
|------------|-------------------------------|
| int        | Index of the element required |

**Returns**

Value corresponding to the index

**void LOWIMacAddress::print ()**

Prints the Mac address in the correct form

**int LOWIMacAddress::setMac (const LOWIMacAddress & rhs)**

Sets the Mac address

| Parameters      | Description    |
|-----------------|----------------|
| LOWIMacAddress& | LOWIMacAddress |

**int LOWIMacAddress::setMac (const unsigned char \*const pAddr)**

Sets the Mac address

| Parameters | Description       |
|------------|-------------------|
| unsigned   | char* Mac address |

**int LOWIMacAddress::setMac (const int addr\_hi24, const int addr\_lo24)**

Sets the Mac address

| Parameters | Description                      |
|------------|----------------------------------|
| int        | Upper 24 bits of the mac address |
| int        | Lower 24 bits of the mac address |

**Member data documentation****const char \*const LOWIMacAddress::TAG = "LOWIMacAddress" [static]**

Log tag

**5.9.4.13 qc\_loc\_fw::LOWIMeasurementInfo Struct Reference****Detailed description**

This struct defines the Measurement Info per Wifi Node. It contains measurement information for different types of scans. At any given time only some values are valid for the scan type being used.

For a DiscoveryScan only rssi and rssi\_timestamp are valid fields For a Ranging scan rtt, rtt\_timestamp and rssi are valid fields. For dual sided RTT additionally preamble, bw, mcsIdx and bitrate are valid fields.

### Public member functions

- `LOWIMeasurementInfo ()`
- `LOWIMeasurementInfo`

### Public attributes

- `int32 meas_age`
- `int32 rtt`
- `int32 rtt_ps`
- `int64 rtt_timestamp`
- `int64 rssi_timestamp`
- `int16 rssi`
- `uint16 reserved1`
- `uint32 reserved2`
- `uint32 tx_bitrate`
- `uint8 tx_preamble`
- `uint8 tx_nss`
- `uint8 tx_bw`
- `uint8 tx_mcsIdx`
- `uint32 rx_bitrate`
- `uint8 rx_preamble`
- `uint8 rx_nss`
- `uint8 rx_bw`
- `uint8 rx_mcsIdx`

### Constructor and destructor documentation

#### `LOWIMeasurementInfo::LOWIMeasurementInfo ()`

`LOWIMeasurementInfo.`

constructor

## Member data documentation

### **int32 qc\_loc\_fw::LOWIMeasurementInfo::meas\_age**

Measurement age msec In units of 1 msec, -1 means info not available

### **uint16 qc\_loc\_fw::LOWIMeasurementInfo::reserved1**

Reserved

### **int16 qc\_loc\_fw::LOWIMeasurementInfo::rssI**

Signal strength in 0.5dBm increments.

### **int64 qc\_loc\_fw::LOWIMeasurementInfo::rssI\_timestamp**

Measurement time stamp in milliseconds. For DiscoveryScanResponse time stamp is corresponding to when the beacon was received. Not applicable to APs

### **int32 qc\_loc\_fw::LOWIMeasurementInfo::rtt**

RTT - value in nsec. 0 is considered to be an invalid rtt value.

### **int32 qc\_loc\_fw::LOWIMeasurementInfo::rtt\_ps**

RTT - value in pico sec. 0 is considered to be invalid rtt

### **uint32 qc\_loc\_fw::LOWIMeasurementInfo::rx\_bitrate**

RX Parameters bitrate in units of 100 Kbps

### **uint8 qc\_loc\_fw::LOWIMeasurementInfo::rx\_bw**

Bandwidth. Refer to Table 2

### **uint8 qc\_loc\_fw::LOWIMeasurementInfo::rx\_mcsIdx**

Modulation Coding Scheme (MCS) index defines the following:

- The number of streams used for Tx & Rx
- the type of modulation used
- the coding rate

**NOTE** Do not apply to legacy frames (Frames using schemes prior to 802.11n)

### **uint8 qc\_loc\_fw::LOWIMeasurementInfo::rx\_nss**

Number of spatial streams

**uint8 qc\_loc\_fw::LOWIMeasurementInfo::rx\_preamble**

Preamble

**uint32 qc\_loc\_fw::LOWIMeasurementInfo::tx\_bitrate**

TX Parameters bitrate

**uint8 qc\_loc\_fw::LOWIMeasurementInfo::tx\_bw**

Bandwidth

**uint8 qc\_loc\_fw::LOWIMeasurementInfo::tx\_mcsIdx**

MCS index defines the following:

- The number of streams used for Tx & Rx
- the type of modulation used
- the coding rate

**NOTE** Does not apply to legacy frames (Frames using schemes prior to 802.11n)

**uint8 qc\_loc\_fw::LOWIMeasurementInfo::tx\_nss**

Number of spatial streams

**uint8 qc\_loc\_fw::LOWIMeasurementInfo::tx\_preamble**

preamble

#### 5.9.4.14 qc\_loc\_fw::LOWIMsapInfo Struct Reference

##### Detailed description

This struct defines the MSAP (Mobility Service Advertisement Protocol) related data. Not applicable to APs.

##### Public attributes

- uint8 protocolVersion
- uint32 venueHash
- uint8 serverIdx

#### 5.9.4.15 qc\_loc\_fw::LOWINodeInfo Struct Reference

Inherited by qc\_loc\_fw::LOWIPeriodicNodeInfo.

## Detailed description

Defines the information for a Wi-Fi node.

### Public member functions

- LOWINodeInfo()
- void validate()

### Public attributes

- LOWIMacAddress bssid
- uint32 frequency
- uint32 band\_center\_freq1
- uint32 band\_center\_freq2
- eNodeType nodeType
- LOWIMacAddress spoofMacId
- eRttType rttType
- eRangingBandwidth bandwidth
- uint32 ftmRangingParameters
- eRangingPreamble preamble
- uint8 num\_pkts\_per\_meas
- uint8 num\_retries\_per\_meas

### Constructor and destructor documentation

#### **LOWINodeInfo::LOWINodeInfo ()**

Constructor

### Member function documentation

#### **void LOWINodeInfo::validate ()**

Validates the NodeInfo

### Member data documentation

#### **uint32 qc\_loc\_fw::LOWINodeInfo::band\_center\_freq1**

Frequency of the center of total bandwidth

#### **uint32 qc\_loc\_fw::LOWINodeInfo::band\_center\_freq2**

Frequency of the center of the second 80 MHz Lobe if BW is 80 MHz + 80 MHz

**eRangingBandwidth qc\_loc\_fw::LOWINodeInfo::bandwidth**

Bandwidth to be used

**LOWIMacAddress qc\_loc\_fw::LOWINodeInfo::bssid**

MacId of the node

**uint32 qc\_loc\_fw::LOWINodeInfo::frequency**

Frequency in MHz, which the node is transmitting

**eNodeType qc\_loc\_fw::LOWINodeInfo::nodeType**

Either PEER\_DEVICE or ACCESS\_POINT

**uint8 qc\_loc\_fw::LOWINodeInfo::num\_pkts\_per\_meas**

Number of packets for each RTT measurement

**uint8 qc\_loc\_fw::LOWINodeInfo::num\_retries\_per\_meas**

Maximum number of times a measurement should be retried if the measurement fails (i.e. it yields no results)

**eRangingPreamble qc\_loc\_fw::LOWINodeInfo::preamble**

Preamble to be used eRttType qc\_loc\_fw::LOWINodeInfo::rttType

The type of RTT to be performed

**5.9.4.16 qc\_loc\_fw::LOWIPeriodicNodeInfo struct reference**

Inherits [qc\\_loc\\_fw::LOWINodeInfo](#).

**Detailed description**

Defines the information for a Wi-Fi node and allows for periodic measurements

**Public Member Functions**

- `LOWIPeriodicNodeInfo ()`
- `void validateParams ()`

**Public Attributes**

- `uint8 periodic`
- `uint32 meas_period`
- `uint32 num_measurements`

## Constructor and destructor documentation

### **LOWIPeriodicNodeInfo::LOWIPeriodicNodeInfo ()**

Constructor

## Member function documentation

### **void LOWIPeriodicNodeInfo::validateParams ()**

This function checks the parameters for every Wi-Fi node in the request and ensures that they are correct. If not, it assigns a valid default parameter so the request can be serviced.

## Member data documentation

### **uint32 qc\_loc\_fw::LOWIPeriodicNodeInfo::meas\_period**

For periodic requests, this indicates the periodicity of the measurements in milliseconds (i.e. every 500 ms, or every 800 ms, etc)

### **uint32 qc\_loc\_fw::LOWIPeriodicNodeInfo::num\_measurements**

For periodic requests, this indicates the number of measurements to be carried out at the ‘period’ given above.

### **uint8 qc\_loc\_fw::LOWIPeriodicNodeInfo::periodic**

indicates whether request for this WiFi node is periodic or one-shot

- 0: one-shot
- non-zero: periodic

## 5.9.4.17 qc\_loc\_fw::LOWIPeriodicRangingScanRequest class reference

Inherits [qc\\_loc\\_fw::LOWIRangingScanRequest](#).

### Detailed description

Class to make a periodic ranging scan request.

### Public member functions

- `vector<LOWIPeriodicNodeInfo> & getNodes ()`
- `LOWIPeriodicRangingScanRequest (uint32 requestId, vector<LOWIPeriodicNodeInfo> &node_info, int64 timestamp)`
- `virtual ~LOWIPeriodicRangingScanRequest ()`
- `virtual eRequestType getRequestType () const`

**Static protected attributes**

- static vector<LOWINodeInfo> emptyNodeInfo

**Constructor and destructor documentation**

**LOWIPeriodicRangingScanRequest::LOWIPeriodicRangingScanRequest (uint32 *requestId*, vector<LOWIPeriodicNodeInfo> & *node\_info*, int64 *timestamp*)**

Constructor

| Parameters                   | Description                                                                                                    |
|------------------------------|----------------------------------------------------------------------------------------------------------------|
| uint32                       | Request id generated by the client This will be echoed back in the corresponding response.                     |
| vector<LOWIPeriodicNodeInfo> | Dynamic array containing wifi nodes that need to be scanned.                                                   |
| int64                        | Timestamp at which the request will be dropped if not processed already. Should be 0 if this is to be ignored. |

**LOWIPeriodicRangingScanRequest::~LOWIPeriodicRangingScanRequest () [virtual]**

Destructor

**Member function documentation**

**vector<LOWIPeriodicNodeInfo> & LOWIPeriodicRangingScanRequest::getNodes ()**

Returns the dynamic array containing the LOWINodeInfo

**Returns**

Dynamic array containing the LOWINodeInfo

LOWIRequest::eRequestType

**LOWIPeriodicRangingScanRequest::getRequestType () const [virtual]**

Returns the request type

**Returns**

eRequestType type of the request

Reimplemented from qc\_loc\_fw::LOWIRangingScanRequest

## Member data documentation

`vector< LOWINodeInfo >`  
`LOWIPeriodicRangingScanRequest::emptyNodeInfo [static], [protected]`

The empty vector ‘nodes’ is passed to the LOWIRangingScanRequest constructor when creating a LOWIPeriodicRangingScanRequest type. That way, LOWIPeriodicRangingScanRequest types can be created using the already existing constructors.

### 5.9.4.18 qc\_loc\_fw::LOWIRangingScanRequest Class Reference

Inherits qc\_loc\_fw::LOWIRequest.

Inherited by qc\_loc\_fw::LOWIPeriodicRangingScanRequest.

#### Detailed description

Class to make a ranging scan request.

#### Public member functions

- `vector< LOWINodeInfo > & getNodes ()`
- `void setTimeoutTimestamp (int64 timestamp)`
- `int64 getTimeoutTimestamp () const`
- `void setReportType (eRttReportType report_type)`
- `eRttReportType getReportType () const`
- `LOWIRangingScanRequest (uint32 requestId, vector < LOWINodeInfo > &node_info, int64 timestamp)`
- `virtual ~LOWIRangingScanRequest ()`
- `virtual eRequestType getRequestType () const`

#### Constructor and destructor documentation

**LOWIRangingScanRequest::LOWIRangingScanRequest (uint32 requestId, vector< LOWINodeInfo > & node\_info, int64 timestamp)**

Constructor

| Parameters           | Description                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------|
| uint32               | Request id generated by the client This will be echoed back in the corresponding response.                                     |
| vector<LOWINodeInfo> | Dynamic array containing wifi nodes that need to be scanned.                                                                   |
| int64                | Absolute system timestamp at which the request will be dropped if not processed already. Should be 0 if this is to be ignored. |

**LOWIRangingScanRequest::~LOWIRangingScanRequest ()[virtual]**

Destructor

**Member function documentation****vector< LOWINodeInfo > & LOWIRangingScanRequest::getNodes ()**

Returns the dynamic array containing the LOWINodeInfo

**Returns**

Dynamic array containing the LOWINodeInfo

**eRttReportType LOWIRangingScanRequest::getReportType () const**

Returns the RTT report type for this request

**Returns**

eRttReportType RTT Report type

**LOWIRequest::eRequestType LOWIRangingScanRequest::getRequestType () const[virtual] const[virtual]**

Returns the request type

**Returns**

eRequestType type of the Request

Implements qc\_loc\_fw::LOWIRequest.

Reimplemented in qc\_loc\_fw::LOWIPeriodicRangingScanRequest.

**int64 LOWIRangingScanRequest::getTimeoutTimestamp () const**

Returns the timestamp at which the request expires

**Returns**

timestamp at which the request expires

**void LOWIRangingScanRequest::setReportType (eRttReportType *report\_type*)**

Sets the RTT report type for the request

**Returns**

void

**void LOWIRangingScanRequest::setTimeoutTimestamp (int64 timestamp)**

Sets the time-out timestamp for the request

**Returns**

void

**5.9.4.19 qc\_loc\_fw::LOWIRangingScanResponse class reference**

Inherits [qc\\_loc\\_fw::LOWIResponse](#).

**Detailed description**

Response to the ranging scan request

**Public member functions**

- [LOWIRangingScanResponse \(uint32 requestId\)](#)
- virtual [~LOWIRangingScanResponse \(\)](#)
- virtual [eResponseType getResponseType \(\)](#)

**Public attributes**

- eScanStatus [scanStatus](#)
- vector<[LOWIScanMeasurement](#) \*> [scanMeasurements](#)

**Constructor and destructor documentation****LOWIRangingScanResponse::LOWIRangingScanResponse (uint32 requestId)**

Constructor

| Parameters | Description                                    |
|------------|------------------------------------------------|
| uint32     | Request Id generated by the client for Request |

**LOWIRangingScanResponse::~LOWIRangingScanResponse ()[virtual]**

Destructor

**Member function documentation****[LOWIResponse::eResponseType](#)****[LOWIRangingScanResponse::getResponseType \(\)\[virtual\]](#)**

Returns the response type

**Returns**

eResponseType type of response

Implements `qc_loc_fw::LOWIRangingScanResponse::scanMeasurements`.

**Member data documentation**

**`vector<LOWIScanMeasurement*> qc_loc_fw::LOWIRangingScanResponse::scanMeasurements`**

Dynamic array containing received ScanMeasurement

**`eScanStatus qc_loc_fw::LOWIRangingScanResponse::scanStatus`**

Status of the scan

**5.9.4.20 qc\_loc\_fw::LOWIRequest class reference**

Inherited by `qc_loc_fw::LOWIAsyncDiscoveryScanResultRequest`, `qc_loc_fw::LOWICacheResetRequest`, `qc_loc_fw::LOWICancelRangingScanRequest`, `qc_loc_fw::LOWICapabilityRequest`, `qc_loc_fw::LOWIDiscoveryScanRequest`, `qc_loc_fw::LOWIFTMRangingRequest`, `qc_loc_fw::LOWINeighborReportRequest`, `qc_loc_fw::LOWIRangingScanRequest`, `qc_loc_fw::LOWISendLCIRequest`, `qc_loc_fw::LOWISetLCIlocationInformation`, and `qc_loc_fw::LOWISetLCRLocationInformation`.

**Detailed description**

Base class for all requests LOWI can handle

**Public types**

```
enum eRequestType {DISCOVERY_SCAN = 0, RANGING_SCAN, CAPABILITY, RESET_CACHE, ASYNC_DISCOVERY_SCAN_RESULTS, PERIODIC_RANGING_SCAN, CANCEL_RANGING_SCAN, SET_LCI_INFORMATION, SET_LCR_INFORMATION, NEIGHBOR_REPORT, SEND_LCI_REQUEST, FTM_RANGE_REQ, LOWI_INTERNAL_MESSAGE}
```

**Public member functions**

- `LOWIRequest (uint32 requestId)`
- `virtual ~LOWIRequest ()=0`
- `LOWIRequest (const LOWIRequest &rhs)`
- `LOWIRequest & operator= (const LOWIRequest &rhs)`
- `const char * getRequestOriginator () const`
- `void setRequestOriginator (const char *const request_originator)`
- `uint32 getRequestID () const`

- virtual eRequestType getRequestType () const =0

### Static public attributes

- static const char \*const TAG = "LOWIRequest"

### Member function documentation

#### **enum qc\_loc\_fw::LOWIRequest::eRequestType**

Type of request

Defines the status of the request.

| Enumerator                          |                                                                                     |
|-------------------------------------|-------------------------------------------------------------------------------------|
| <b>DISCOVERY_SCAN</b>               | Request a passive or active scan from the wifi driver; not Applicable to APs        |
| <b>RANGING_SCAN</b>                 | Request to do RTT ranging with another wifi node                                    |
| <b>CAPABILITY</b>                   | Retrieve LOWI capabilities for ranging, etc.                                        |
| <b>RESET_CACHE</b>                  | Clear the WiFi node cached by LOWI                                                  |
| <b>ASYNC_DISCOVERY_SCAN_RESULTS</b> | Receive any scan results if and when available; not applicable to APs               |
| <b>PERIODIC_RANGING_SCAN</b>        | Do periodic RTT ranging measurements with LOWI managing the periodic requests to FW |
| <b>CANCEL_RANGING_SCAN</b>          | Cancel a PERIODIC_RANGING_SCAN request previously sent                              |
| <b>SET_LCI_INFORMATION</b>          | Set LCI location information                                                        |
| <b>SET_LCR_INFORMATION</b>          | Set LCR location information                                                        |
| <b>NEIGHBOR_REPORT</b>              | Request neighbor report from associated AP; not applicable to APs                   |
| <b>SEND_LCI_REQUEST</b>             | Request LCI of remote STA                                                           |
| <b>FTM_RANGE_REQ</b>                | Request FTMRange to remote STA                                                      |
| <b>LOWI_INTERNAL_MESSAGE</b>        | LOWI internal message                                                               |

### Constructor and destructor documentation

#### **LOWIRequest::LOWIRequest (uint32 requestId)**

Constructor

| Parameters | Description                        |
|------------|------------------------------------|
| uint32     | Request Id generated by the client |

#### **LOWIRequest::~LOWIRequest ()[pure virtual]**

Destructor

**LOWIRequest::LOWIRequest (const LOWIRequest & rhs)**

Copy Constructor

**Member function documentation****uint32 LOWIRequest::getRequestId () const**

Returns the RequestId

**Returns**

RequestId generated by the user of the API

**const char \* LOWIRequest::getRequestOriginator () const**

Returns the request originator

**Returns**

char\* request originator's id

**virtual eRequestType qc\_loc\_fw::LOWIRequest::getRequestType () const [pure virtual]**

Returns the request type

**Returns**

eRequestType type of request

Implemented in qc\_loc\_fw::LOWIFTMRangingRequest, qc\_loc\_fw::LOWISendLCIRequest, qc\_loc\_fw::LOWINeighborReportRequest, qc\_loc\_fw::LOWISetLCRLocationInformation, qc\_loc\_fw::LOWISetLCILocationInformation, qc\_loc\_fw::LOWICancelRangingScanRequest, qc\_loc\_fw::LOWIPeriodicRangingScanRequest, qc\_loc\_fw::LOWIAsyncDiscoveryScanResultRequest, qc\_loc\_fw::LOWIRangingScanRequest, qc\_loc\_fw::LOWIDiscoveryScanRequest, qc\_loc\_fw::LOWICacheResetRequest, and qc\_loc\_fw::LOWICapabilityRequest

**LOWIRequest & LOWIRequest::operator= (const LOWIRequest & rhs)**

Assignment operator

**void LOWIRequest::setRequestOriginator (const char \*const request\_originator)**

Sets the request originator for the request. Used internally by LOWI to identify the originator of the request and send the responses back to the originator.

**NOTE** Clients of LOWI API do not need to set the originator as there is no affect of doing so.

## Member data documentation

`const char *const LOWIRequest::TAG = "LOWIRequest"[static]`

Log tag

### 5.9.4.21 qc\_loc\_fw::LOWIResponse Class Reference

Inherited by `qc_loc_fw::LOWICacheResetResponse`, `qc_loc_fw::LOWICapabilityResponse`, `qc_loc_fw::LOWIDiscoveryScanResponse`, `qc_loc_fw::LOWIRangingScanResponse`, and `qc_loc_fw::LOWIStatusResponse`.

#### Detailed description

Base class for response

#### Public types

- enum `eResponseType` { `RESPONSE_TYPE_UNKNOWN` = 0, `DISCOVERY_SCAN`, `RANGING_SCAN`, `CAPABILITY`, `RESET_CACHE`, `ASYNC_DISCOVERY_SCAN_RESULTS`, `LOWI_STATUS` }
- enum `eScanStatus` { `SCAN_STATUS_UNKNOWN` = 0, `SCAN_STATUS_SUCCESS` = 1, `SCAN_STATUS_BUSY` = 2, `SCAN_STATUS_DRIVER_ERROR` = 3, `SCAN_STATUS_DRIVER_TIMEOUT` = 4, `SCAN_STATUS_INTERNAL_ERROR` = 5, `SCAN_STATUS_INVALID_REQ` = 6, `SCAN_STATUS_NOT_SUPPORTED` = 7, `SCAN_STATUS_NO_WIFI` = 8, `SCAN_STATUS_TOO_MANY_REQUESTS` = 9, `SCAN_STATUS_OUT_OF_MEMORY` = 10 }

#### Public Member Functions

- `LOWIResponse`(`uint32 requestId`)
- virtual `~LOWIResponse`()=0
- `uint32 getRequestId()`
- virtual `eResponseType getResponseType()`=0

#### Static public attributes

- static const `char *const TAG` = ‘LOWIResponse’

#### Member function documentation

##### enum qc\_loc\_fw::LOWIResponse::eResponseType

Type of response; defines the status of the request

| Enumerator                         |                                                                    |
|------------------------------------|--------------------------------------------------------------------|
| <code>RESPONSE_TYPE_UNKNOWN</code> | Unknown response type                                              |
| <code>DISCOVERY_SCAN</code>        | Response carrying the results produced by a DISCOVERY_SCAN request |

|                                     |                                                                                            |
|-------------------------------------|--------------------------------------------------------------------------------------------|
| <b>RANGING_SCAN</b>                 | Response carrying the results produced by a RANGING_SCAN request                           |
| <b>CAPABILITY</b>                   | List of capabilities supported by LOWI                                                     |
| <b>RESET_CACHE</b>                  | Response to a RESET_CACHE request                                                          |
| <b>ASYNC_DISCOVERY_SCAN_RESULTS</b> | Response carrying asynchronous results produced by an ASYNC_DISCOVERY_SCAN_RESULTS request |
| <b>LOWI_STATUS</b>                  | Status response                                                                            |

**enum qc\_loc\_fw::LOWIResponse::eScanStatus**

Defines status of the scan request.

| Enumerator                           |                                                                                                                                                                                                                                                  |
|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SCAN_STATUS_SUCCESS</b>           | Measurements were obtained successfully from the WLAN driver. Note that SUCCESS does not guarantee that there are any measurements in this packet. It is possible to have zero measurements and a SUCCESS; if there were no APs in the vicinity. |
| <b>SCAN_STATUS_BUSY</b>              | Indicates that the number of pending clients have reached the maximum                                                                                                                                                                            |
| <b>SCAN_STATUS_DRIVER_ERROR</b>      | Unable to initiate request to driver                                                                                                                                                                                                             |
| <b>SCAN_STATUS_DRIVER_TIMEOUT</b>    | Unable to get response from driver                                                                                                                                                                                                               |
| <b>SCAN_STATUS_INTERNAL_ERROR</b>    | There is an internal error condition that prevents LOWI from providing any measurements                                                                                                                                                          |
| <b>SCAN_STATUS_INVALID_REQ</b>       | Invalid request                                                                                                                                                                                                                                  |
| <b>SCAN_STATUS_NOT_SUPPORTED</b>     | Request not supported                                                                                                                                                                                                                            |
| <b>SCAN_STATUS_NO_WIFI</b>           | Wi-Fi not enabled                                                                                                                                                                                                                                |
| <b>SCAN_STATUS_TOO_MANY_REQUESTS</b> | Too many instances of this request type                                                                                                                                                                                                          |
| <b>SCAN_STATUS_OUT_OF_MEMORY</b>     | Out of memory condition                                                                                                                                                                                                                          |

**Constructor and destructor documentation****LOWIResponse::LOWIResponse (uint32 requestId)**

Constructor

| Parameters | Description                                   |
|------------|-----------------------------------------------|
| uint32     | requestId generated by the client for request |

**LOWIResponse::~LOWIResponse ()[pure virtual]**

Destructor

## Member function documentation

### `uint32 LOWIResponse::getRequestId ()`

Request id generated and provided in request by the client. Echoed back in the response.

#### Returns

Corresponding request id

### `virtual eResponseType qc_loc_fw::LOWIResponse::getResponseType () [pure virtual]`

Returns the response type

#### Returns

eResponseType Type of Response

Implemented in `qc_loc_fw::LOWIStatusResponse`, `qc_loc_fw::LOWIAsyncDiscoveryScanResultResponse`, `qc_loc_fw::LOWIRangingScanResponse`, `qc_loc_fw::LOWIDiscoveryScanResponse`, `qc_loc_fw::LOWICacheResetResponse`, and `qc_loc_fw::LOWICapabilityResponse`.

## Member data documentation

### `const char *const LOWIResponse::TAG = "LOWIResponse" [static]`

Log tag

## 5.9.4.22 qc\_loc\_fw::LOWIScanMeasurement class reference

### Detailed description

This class defines the measurement taken for every scan request. This contains the measurements corresponding to discovery / ranging scan requests. However, the fields are valid / invalid based on type of scan as follows.

### Public types

- enum `eTargetStatus` {
 `LOWI_TARGET_STATUS_SUCCESS` = 0,  
`LOWI_TARGET_STATUS_FAILURE` = 1,  
`LOWI_TARGET_STATUS_RTT_FAIL_NO_RSP` = 2,  
`LOWI_TARGET_STATUS_RTT_FAIL_REJECTED` = 3,  
`LOWI_TARGET_STATUS_RTT_FAIL_FTM_TIMEOUT` = 4,  
`LOWI_TARGET_STATUS_RTT_TARGET_ON_DIFF_CHANNEL` = 5,  
`LOWI_TARGET_STATUS_RTT_FAIL_TARGET_NOT_CAPABLE` = 6,
 }

```
    LOWI_TARGET_STATUS_RTT_FAIL_INVALID_TS = 7,  
    LOWI_TARGET_STATUS_RTT_FAIL_TARGET_BUSY_TRY_LATER = 8,  
    LOWI_TARGET_STATUS_MAX}
```

### Public member functions

- LOWIScanMeasurement()
- ~LOWIScanMeasurement()
- LOWIScanMeasurement(const LOWIScanMeasurement &rhs)
- LOWIScanMeasurement & operator=(const LOWIScanMeasurement &rhs)

### Public attributes

- LOWIMacAddress bssid
- uint32 frequency
- uint32 band\_center\_freq1
- uint32 band\_center\_freq2
- uint32 info
- uint32 tsfDelta
- uint32 ranging\_features\_supported
- uint64 rttMeasTimeStamp
- bool associatedToAp
- bool isSecure
- eNodeType type
- eRttType rttType
- LOWISsid ssid
- LOWIMsapInfo \* msapInfo
- int8 cellPowerLimitdBm
- uint8 indoor\_outdoor
- vector<LOWIMeasurementInfo \* > measurementsInfo
- eTargetStatus targetStatus
- uint8 country\_code [LOWI\_COUNTRY\_CODE\_LEN]
- uint32 measurementNum
- uint16 beaconPeriod
- uint16 beaconCaps
- vector<int8 > ieData
- uint16 num\_frames\_attempted

- uint16 actual\_burst\_duration
- uint8 negotiated\_num\_frames\_per\_burst
- uint8 retry\_after\_duration
- uint8 negotiated\_burst\_exp
- LOWILocationIE \* lciInfo
- LOWILocationIE \* lcrInfo
- uint32 location\_features\_supported

### Static public attributes

- static const char \*const TAG = "LOWIScanMeasurement"

### Member function documentation

#### **enum qc\_loc\_fw::LOWIScanMeasurement::eTargetStatus**

This is an enumeration of the list of error codes the Wi-Fi driver will send to LOWI controller with the scan measurements on a per target basis.

### Constructor and destructor documentation

#### **LOWIScanMeasurement::LOWIScanMeasurement ()**

Constructor

#### **LOWIScanMeasurement::~LOWIScanMeasurement ()**

Destructor

#### **LOWIScanMeasurement::LOWIScanMeasurement (const LOWIScanMeasurement & rhs)**

Copy constructor

### Member function documentation

#### **LOWIScanMeasurement & LOWIScanMeasurement::operator= (const LOWIScanMeasurement & rhs)**

Assignment operator

### Member data documentation

#### **uint16 qc\_loc\_fw::LOWIScanMeasurement::actual\_burst\_duration**

Actual time taken by FW to finish one burst of measurements (unit: ms)

**bool qc\_loc\_fw::LOWIScanMeasurement::associatedToAp**

Flag indicating if we are associated with this AP. Not Applicable to APs

**uint32 qc\_loc\_fw::LOWIScanMeasurement::band\_center\_freq1**

Operating Channel Information

**uint16 qc\_loc\_fw::LOWIScanMeasurement::beaconCaps**

Capabilities advertised in the beacon

**uint16 qc\_loc\_fw::LOWIScanMeasurement::beaconPeriod**

The following four params – beaconPeriod – beaconCaps – ieLen – ieData are part of the results for background scansPeriod advertised in the beacon

**LOWIMacAddress qc\_loc\_fw::LOWIScanMeasurement::bssid**

BSSID of the Wi-Fi node

**int8 qc\_loc\_fw::LOWIScanMeasurement::cellPowerLimitdBm**

Cell power limit in dBm. Only valid for discovery scan results, if available. For ranging scan results will be always 0.

**uint8 qc\_loc\_fw::LOWIScanMeasurement::country\_code[LOWI\_COUNTRY\_CODE\_LEN]**

Contains the Country Code for example, 'U' 'S' if there is no country code found, then the array will contain 0 0

**uint32 qc\_loc\_fw::LOWIScanMeasurement::frequency**

Operating Channel Information

**vector<int8> qc\_loc\_fw::LOWIScanMeasurement::ieData**

Blob of all the information elements found in the beacon

**uint8 qc\_loc\_fw::LOWIScanMeasurement::indoor\_outdoor**

Indicates if AP is indoor or outdoor '' indicates - Do not care 'T' indicates - Indoor 'O' indicates - Outdoor

**uint32 qc\_loc\_fw::LOWIScanMeasurement::info**

The following info field is a bit field that contains PHY mode and flags for this bssid. Bits 0-6 contain the PHY mode Bits 7-13 contain the flags

**bool qc\_loc\_fw::LOWIScanMeasurement::isSecure**

Secure access point or not. Only valid for DiscoveryScanResponse

**LOWILocationIE\* qc\_loc\_fw::LOWIScanMeasurement::lcilInfo**

LCI information element

**LOWILocationIE\* qc\_loc\_fw::LOWIScanMeasurement::lcrInfo**

LCR information element

**uint32 qc\_loc\_fw::LOWIScanMeasurement::location\_features\_supported**

bit mask used to store which location features are supported by the AP

**uint32 qc\_loc\_fw::LOWIScanMeasurement::measurementNum**

Measurement number. In case of periodic ranging scan measurements, this will provide a counter that the client can use to track the number of measurements at any given point during the ranging request. It does not apply to single-shot ranging requests. For single-shot requests this will always be zero.

**vector<LOWIMeasurementInfo\*> qc\_loc\_fw::LOWIScanMeasurement::measurementsInfo**

Dynamic array containing measurement info per Wi-Fi node. DiscoveryScan and background scans have one measurement whereas the vector can contain multiple MeasurementInfo for a ranging scan.

**LOWIMsapInfo\* qc\_loc\_fw::LOWIScanMeasurement::msapInfo**

MsapInfo - valid if not NULL. Only valid for DiscoveryScanResponse

**uint8 qc\_loc\_fw::LOWIScanMeasurement::negotiated\_burst\_exp**

Number of "FTM bursts" negotiated with peer/target. This is indicated in the form of an exponent. The number of bursts =  $2^{\text{negotiated\_burst\_exp}}$

**uint8 qc\_loc\_fw::LOWIScanMeasurement::negotiated\_num\_frames\_per\_burst**

Number of "FTM frames per burst" negotiated with peer/target.

**uint16 qc\_loc\_fw::LOWIScanMeasurement::num\_frames\_attempted**

Total RTT measurement Frames attempted

**uint8 qc\_loc\_fw::LOWIScanMeasurement::retry\_after\_duration**

If Target/peer fails to accept an FTM session. Peer provides when it to retry FTM session. this field has the time after which FTM session can be retried. uint: seconds

**uint64 qc\_loc\_fw::LOWIScanMeasurement::rttMeasTimeStamp**

Timestamp of when RTT measurement was taken - applies only to RTT measurements at this time

**eRttType qc\_loc\_fw::LOWIScanMeasurement::rttType**

Type of RTT measurement performed, if applicable

**LOWISsid qc\_loc\_fw::LOWIScanMeasurement::ssid**

SSID. Only valid for DiscoveryScanResponse

**const char \*const LOWIScanMeasurement::TAG =  
"LOWIScanMeasurement" [static]**

Log tag

**eTargetStatus qc\_loc\_fw::LOWIScanMeasurement::targetStatus**

Status for the measurements associated with this Target.

**eNodeType qc\_loc\_fw::LOWIScanMeasurement::type**

Type of the Wi-Fi Node. Only valid for DiscoveryScanResponse

### 5.9.4.23 qc\_loc\_fw::LOWISendLCIRequest Class Reference

Inherits qc\_loc\_fw::LOWIRequest.

#### Detailed description

LCI request used by AP to request LCI from remote STA or AP

#### Public member functions

- LOWISendLCIRequest (uint32 requestId, LOWIMacAddress bssid)
- virtual ~LOWISendLCIRequest ()
- const LOWIMacAddress & getBssid () const
- virtual eRequestType getRequestType () const

## Constructor and destructor documentation

**LOWISendLCIRequest::LOWISendLCIRequest (uint32 requestId, LOWIMacAddress bssid)**

Constructor

| Parameters     | Description                                                                                |
|----------------|--------------------------------------------------------------------------------------------|
| uint32         | Request id generated by the client This will be echoed back in the corresponding response. |
| LOWIMacAddress | Bssid of target STA or AP                                                                  |

**LOWISendLCIRequest::~LOWISendLCIRequest ()[virtual]**

Destructor

## Member function documentation

**const LOWIMacAddress & LOWISendLCIRequest::getBssid () const**

Returns the bssid of target STA or AP for this request

### Returns

LOWIMacAddress &

**LOWIRequest::eRequestType LOWISendLCIRequest::getRequestType () const[virtual]**

Returns the request type

### Returns

eRequestType type of request

Implements qc\_loc\_fw::LOWIRequest.

## 5.9.4.24 qc\_loc\_fw::LOWISetLCILocationInformation class reference

Inherits qc\_loc\_fw::LOWIRequest.

### Detailed description

Set LCI location information

### Public member functions

- LOWISetLCILocationInformation (uint32 requestID, LOWILciInformation &lciInfo, uint32 &usageRules)
- ~LOWISetLCILocationInformation ()
- const LOWILciInformation & getLciParams () const

- `uint32 getUsageRules () const`
- `virtual eRequestType getRequestType () const` Additional Inherited Members

### **Constructor and destructor documentation**

**LOWISetLCILocationInformation::LOWISetLCILocationInformation (uint32 *requestID*, LOWILcInformation & *lciInfo*, uint32 & *usageRules*)**

Constructor

| Parameters        | Description                                                                                |
|-------------------|--------------------------------------------------------------------------------------------|
| uint32            | Request id generated by the client This will be echoed back in the corresponding response. |
| LOWILcInformation | LCI information to be set                                                                  |
| uint32            | usageRules as per defined in 802.11mc standard                                             |

**LOWISetLCILocationInformation::~LOWISetLCILocationInformation ()**

Destructor

### **Member function documentation**

**const LOWILcInformation & LOWISetLCILocationInformation::getLciParams ()  
const**

Returns LCI information injected into device

**LOWIRequest::eRequestType LOWISetLCILocationInformation::getRequestType ()  
const[virtual]**

Returns the request type

#### **Returns**

eRequestType type of the Request

Implements qc\_loc\_fw::LOWIRequest.

**uint32 LOWISetLCILocationInformation::getUsageRules () const**

Returns the usage rules variable as per defined in 802.11mc standard

### **5.9.4.25 qc\_loc\_fw::LOWISetLCRLocationInformation class reference**

Inherits qc\_loc\_fw::LOWIRequest.

#### **Detailed description**

Set LCR location information

### Public member functions

- `LOWISetLCRLocationInformation(uint32 requestID, LOWILcrInformation &lcrInfo)`
- `~LOWISetLCRLocationInformation()`
- `const LOWILcrInformation & getLcrParams() const`
- `virtual eRequestType getRequestType() const`

### Constructor and destructor documentation

**`LOWISetLCRLocationInformation::LOWISetLCRLocationInformation(uint32 requestID, LOWILcrInformation & lcrInfo)`**

Constructor lcrInfo: lcrInformation to set

**`LOWISetLCRLocationInformation::~LOWISetLCRLocationInformation()`**

Destructor

### Member function documentation

**`const LOWILcrInformation & LOWISetLCRLocationInformation::getLcrParams() const`**

Returns LCR information injected into device

**`LOWIRequest::eRequestType LOWISetLCRLocationInformation::getRequestType() const[virtual]`**

Returns the request type

### Returns

`eRequestType` type of the request

Implements `qc_loc_fw::LOWIRequest`.

## 5.9.4.26 qc\_loc\_fw::LOWISsid class reference

### Detailed description

Class for SSID

### Public member functions

- `bool isSSIDValid() const`
- `int setSSID(const unsigned char *const ssid, const int length)`
- `int getSSID(unsigned char *const pSsid, int *const pLength) const`
- `int compareTo(const LOWISsid &ssid)`

**Static public attributes**

- static const char \*const TAG = "LOWISSID"

**Member function documentation****int LOWISSID::compareTo (const LOWISSID & ssid)**

Compares the ssid with the current

| Parameters | Description              |
|------------|--------------------------|
| LOWISSID&  | LOWISSID to compare with |

**Returns**

0 for equal, non zero otherwise

**int LOWISSID::getSSID (unsigned char \*const pSsid, int \*const pLength) const**

Gets SSID

|     |          |                            |
|-----|----------|----------------------------|
| out | unsigned | char* SSID to be retrieved |
| out | int*     | Length of the SSID         |

**Returns**

0 for success, non zero for error

**bool LOWISSID::isSSIDValid () const**

Checks if the SSID is valid

**Returns**

True for valid, false otherwise

**int LOWISSID::setSSID (const unsigned char \*const ssid, const int length)**

Sets SSID

| Parameters | Description          |
|------------|----------------------|
| unsigned   | char* SSID to be set |
| int        | Length of the SSID   |

**Returns**

0 for success, non zero for error

## Member data documentation

```
const char *const LOWISsid::TAG = "LOWISsid";[static]
```

Log tag

### 5.9.5 Configure round-trip time on AP using UCI commands

Wi-fi Location certification program is based on IEEE802.11mc where clients and AP use Fine Timing Measurements (FTMs) to do ranging (also known as RTT - Round trip time). To enable ranging or RTT capability on interface X on AP, following commands can be used:

```
uci set wireless.@wifi-iface[X].enable_rtt='1'
uci set wireless.@wifi-iface[X].enable_lci='1'
uci set wireless.@wifi-iface[X].enable_lcr='1'
uci set wireless.@wifi-iface[X].rrm='1'
uci commit
```

## 5.10 Static MAC ID generation

MAC ID statically generated by specifying the VAP ID in wlanconfig cli command for VAP creation.

This feature works if enable\_macreq is set to 1 for a particular radio.

By default if user request for vap id 0-7 it will be mapped to 8-15 internally. This behavior is untouched to maintained backward compatibility.

If user requests vap id from 8-15 it will be mapped to 0-7. So for vap id 8 MAC address will be same as HW MAC address since vap id 8 will be treated as 0 internally.

By default this feature generates new MAC address by adding HW MAC address MSB's 4th, 5th, 6th and 7th bits of requested VAP ID and setting 2nd bit to indicate locally administered.

However user can enable this feature to work by modifying HW MAC LSB's 0th, 1st, 2nd and 3rd bits, which by default is protected under ATH\_SUPPORT\_VAP\_ID\_LSB macro.

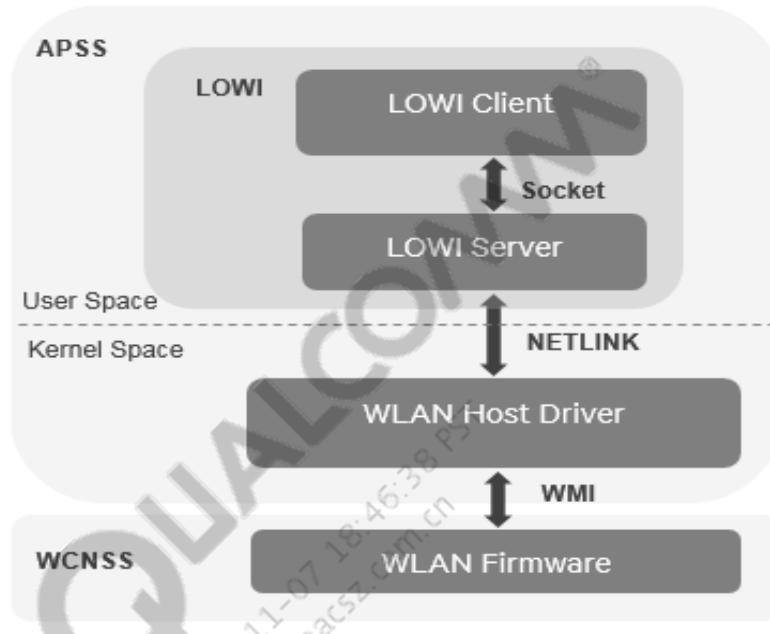
If code will be recompiled by defining ATH\_SUPPORT\_VAP\_ID\_LSB macro then MAC ID and requested VAP ID will have one to one mapping (i.e. 0-15 vap id will be directly used to generate MAC ID) unlike the above mapping.

### Example to create VAP:

```
# iwpriv wifi0 enable_macreq 1
# wlanconfig ath0 create wlandev wifi0 wlanmode ap vapid 0
# wlanconfig ath1 create wlandev wifi0 wlanmode ap vapid 1
...
# wlanconfig ath15 create wlandev wifi0 wlanmode ap vapid 15
```

## 5.11 Positioning: LOWI on AP

Location WLAN Interface (LOWI) is an application module that provides APIs for ranging and RTT (round trip time) measurements, trigger IEEE 802.11mc specific action frames, serialize requests to driver (queues incoming requests). Users can perform ranging measurements using LOWI command line test tool.



**Figure 5-13 Block Diagram of LOWI on Access Point**

Figure 5-13 shows the block diagram of LOWI on AP and how it interacts with HOST driver and Firmware.

There are two components to LOWI user space module: LOWI client and LOWI server. LOWI client is a command line test tool that parses user input and provides it to LOWI server via socket. LOWI server implements APIs to generate LOWI messages for measurements and IEEE 802.11mc frame triggers. LOWI server communicates with HOST driver over Netlink socket.

WLAN HOST driver provides an interface to receive messages from LOWI user space module over Netlink socket. It processes LOWI message based on its type. Some messages are consumed at HOST driver and some are sent to WLAN Firmware. For example: Ranging measurement request is pass through for HOST and goes to Firmware, whereas LOWI message to generate 802.11mc Where Are You (LCI Request) frame is consumed at HOST and corresponding action frame is sent on air by HOST. HOST driver communicates with Firmware over WMI interface. Measurement response from Firmware are received at HOST via WMI event handler and then passed to LOWI user space.

LOWI user space module is agnostic of radio and VAPs underneath. HOST driver finds appropriate radio/VAP based on channel or station mac address in LOWI request frame (for example: radio supporting specific channel, VAP where specific MAC address is associated). It then delivers LOWI frame to appropriate firmware instance.

LOWI on AP is supported only on offload-based radios. Main files and directories related to LOWI user module and HOST driver are listed below:

#### **LOWI user module: under qsdk/qca-lowi**

- internal/lowi/lowi\_client
- internal/lowi/lowi\_server

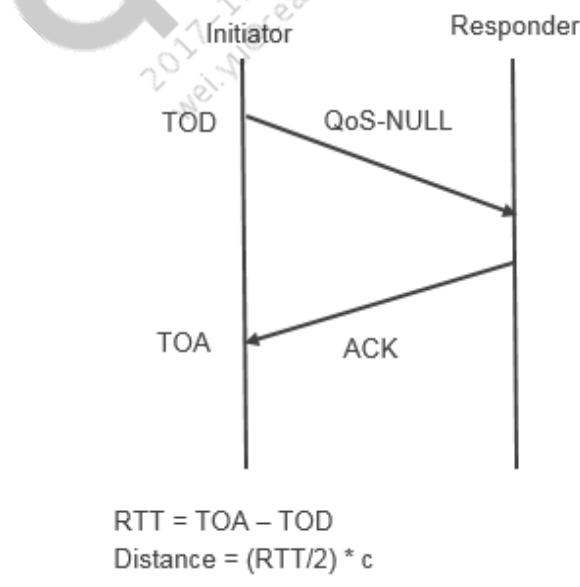
#### **HOST driver interface: under qsdk/qca-wifi-10.4**

- os/linux/src/ath\_lowi\_if.c
- umac/rtt/ieee80211\_rtt.c
- offload/wlan/umac\_offload\_if/ol\_if\_lowi.c

### **5.11.1 RTT Types**

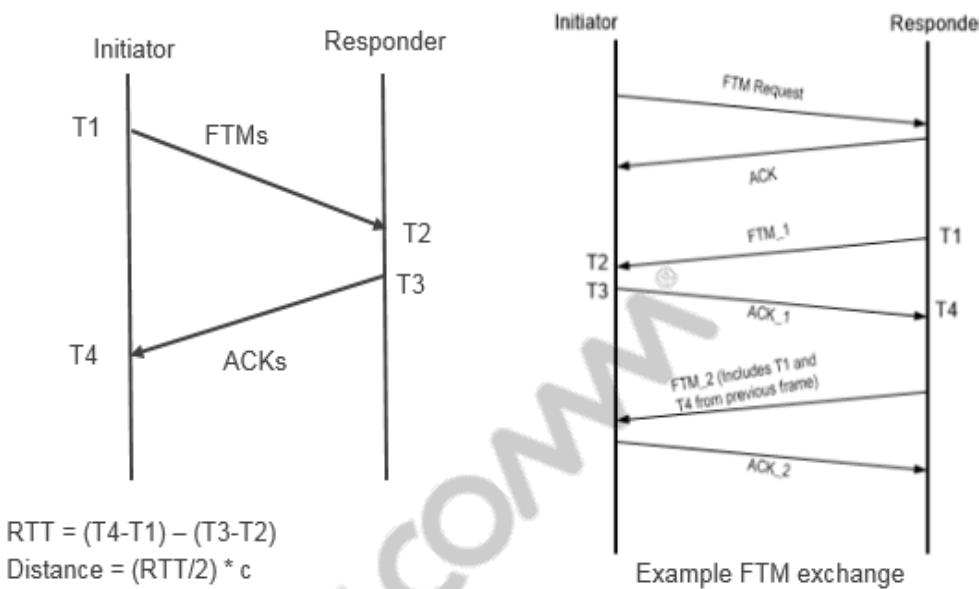
LOWI on AP can be used for RTT (Round Trip Time) measurements and triggering various IEEE 802.11mc action frames. Two types of RTT measurements are supported: Single-sided RTT and Double-sided RTT.

**Single sided RTT:** Responder (Client or another AP) responds with an ACK for transmitted QoS-NUL frames. Timestamps at initiator is collected to compute RTT and distance as shown in Figure 7-11a.



**Figure 5-14 Single Sided RTT example**

**Double sided RTT:** Responder (Client or another AP) is more involved and it follows IEEE 802.11mc protocol (refer to IEEE 802.11mc draft specification) for ranging using Fine Timing Measurement (FTM) frames. Timestamps are used on both sides and RTT and distance is computed as shown in Figure 7-11b.



**Figure 5-15 Double Sided RTT example**

#### 5.11.1.1 LOWI features

LOWI on AP can trigger following types of requests:

- Ranging measurements (Single-sided and Double-sided RTT)
- LCR configuration
- LCI configuration
- Where Are You Request (a.k.a. LCI Request) to connected STA
- FTM Range Request (FTMRR) to connected STA

#### 5.11.2 How to Perform Ranging

Single sided RTT can be done from AP to Client and Client to AP, whereas Double-sided RTT can only be initiated by Client (i.e. Client to AP only). Double sided RTT needs IEEE 802.11mc capable client i.e. supports Fine Timing Measurement (FTM) frames, whereas single sided RTT has no such requirement from client.

In order to run ranging measurements:

1. Start LOWI service
2. Connect two devices that support ranging (AP and client)
3. Create input xml file to provide user input for ranging
4. Run **LOWI-test** console command which uses input xml file to start measurement
5. Collect results (Results are shown on console as well as written in a file on AP)

LOWI service is disabled by default. It can be started either by adding following as new sub-section in /etc/config/wireless file

```
config lowi 'lowi'
option enable '1'
```

OR using following UCI commands

```
uci set wireless.lowi=lowi
uci set wireless.lowi.enable=1
uci commit
```

Execute “wifi” command in order to restart wireless interfaces and apply new LOWI service. When interface comes up, LOWI service running in background can be checked via “ps” command (one should see LOWI process /usr/sbin/lowi-server).

Create input xml file on initiator (for example on Client if running double-sided RTT) and provide ranging details. Example input xml file (ap\_list.xml) is shown below, which was created on WDS Client console (under /tmp) for double-sided ranging at VHT80 with mac address 8C:FD:F0:01:D1:15.

```
<body>
<ranging>
<ap>
<band>1</band> <!-- 0 = 2.4 GHz, 1 = 5 GHz -->
<rttType>3</rttType> <!-- 2 = Single, 3 = Double -->
<numFrames>5</numFrames>
<bw>2</bw> <!-- 0=20, 1=40, 2=80, 3=160 -->
<preamble>2</preamble> <!-- 0=Legacy, 1=HT, 2=VHT -->
<asap>1</asap>
<lci>0</lci>
<civic>0</civic>
<burstsexp>0</burstsexp>
<burstduration>15</burstduration>
<burstperiod>0</burstperiod>
<center_freq1>5775</center_freq1>
<center_freq2>0</center_freq2>
<ch>149</ch>
<mac>8C:FD:F0:01:D1:15</mac>
</ap>
</ranging>
</body>
```

Fields to note in ranging input xml file:

1. rttType: 2 is for single-sided and 3 for double-sided RTT
2. numframes: number of FTM's per burst (max value 25)
3. bw: 0 = 20MHz, 1 = 40MHz, 3 = 80MHz
4. preamble: 0 = Legacy, 1 = HT, 2 = VHT
5. center\_freq1: This will change based on channel and bandwidth (for example, Channel 149-20Hz: 5745, Channel 149-40MHz: 5755, Channel 149-80MHz: 5775)
6. ch: Channel devices are operating mac: Mac address of responder

To trigger measurement request, run the following low-test command

### **lowi-test -r ap\_list.xml -n 3**

Above command will trigger double-sided RTT measurement with 3 measurement iterations. Results are displayed on console as well as stored under /usr/share/location/lowi\_lowi\_ap\_summary.csv. User can provide different file name to store different measurement runs using -s <summary\_file> option with **lowi-test** command. Measurement RTT values are shown in pico seconds. LOWI provides Min, Max and Average values of all the iterations that were run as part of the test.

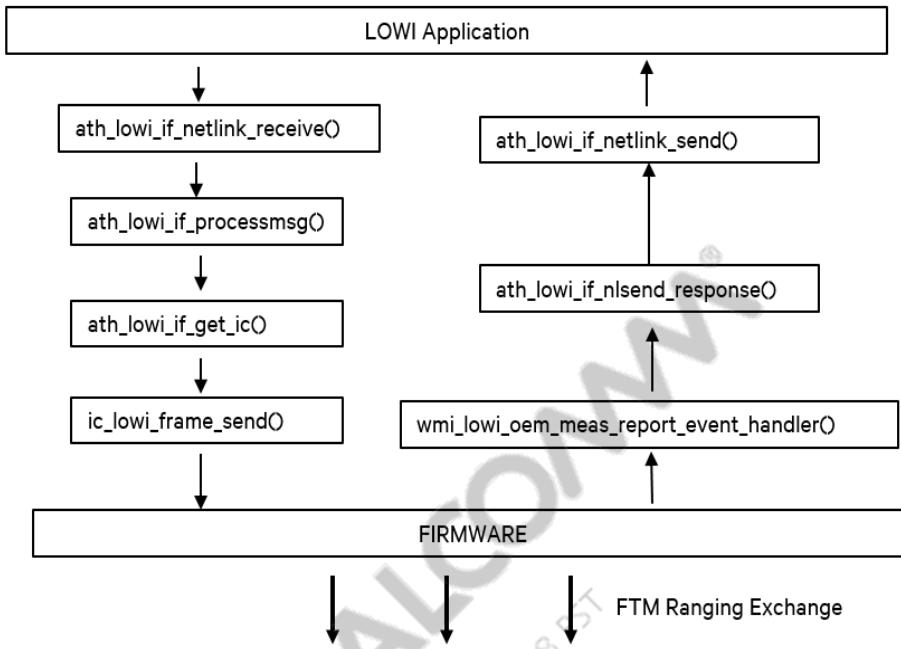
Example result for RTT measurement:

```
STARTING RTT Measurement (1)*****
TIME Read = 1460160568.821491027 = 1460160568821msec
[139416.52][QCALOG-LOWIUTILS] inPostcardToResponse - FROM: LOWI-SERVER,
TO: 1074631632-LOWICLIENT, RESP: LOWI_RANGING_SCAN
==== Ranging SCAN RESULTS ===
[1]itr 8c:fd:f0:01:d1:15 149 555100 -80 2016/4/9:0:9:28:844
[1]itr 8c:fd:f0:01:d1:15 149 476900 -82 2016/4/9:0:9:28:845
==== RANGING SCAN RESULTS END (1 APs Found) ===
TIME Read = 1460160568.847545385 = 1460160568847msec
RTT Measurement request (1 of 1)- SUCCESS, Rsp time 26[139416.53][QCALOG-
LOWICLIENTRECEIVER] ~LOWICLIENTRECEIVER - After join complete
Issued scan Type: RANGING
Summary stats: Scan Type: RANGING
Avg Response Time: Ranging: 26 ms
          AP      Chan   Detection rate      RSSI(dBm)           RTT(psec)
                         Min    Max Avg(dBm) Avg(W)     Min    Max
Avg
8c:fd:f0:01:d1:15 149      1/1   (100%)    -41     -40     -40  476900  555100
516000
```

Timestamp value of pico seconds can be converted to distance in meters by multiplying it by c (speed of light) and divide by 2. For triggering other types of LOWI request, for example, FTMRR or Where Are You (LCI request) frame, please refer to LOWI test tool: User's Guide document.

### **5.11.3 Example Call Flow in HOST Driver**

When a message is received from LOWI application at HOST driver, it is handled by `ath_lowi_if_netlink_receive()` which then calls `ath_lowi_if_processmsg()`, which process messages based on its type. In order to send me LOWI messages to correct FW instance, correct radio is selected based on either first radio with enabled VAP or radio which supports specific channel or radio where a specific mac address station is currently connected. Once appropriate radio is obtained, based on LOWI message type, message is forwarded to either FW using `ol_if_lowi.c` functions or consumed at Host using `ieee80211_rtt.c` functions.



**Figure 5-16 Example call flow in HOST driver for Measurement request/response**

#### 5.11.4 Supported LOWI Message Types

LOWI messages received by Host driver have ANI message header in addition to payload. ANI header have two fields: **Message Type** and **Message Length**. This header is striped before sending message to Firmware. Similarly, ANI header is added by Host driver while sending messages back to LOWI. Following table shows messages currently supported between LOWI application and Host driver.

Message ID	Value	Description	To/From Module
ANI_MSG_APP_REG_REQ	0x1	APP registration request message	To Host Driver
ANI_MSG_APP_REG_RSP	0x2	APP registration response message	From Host Driver
ANI_MSG_OEM_DATA_REQ	0x3	OEM/RTT command message	To FW
ANI_MSG_OEM_DATA_RSP	0x4	OEM/RTT report message	From FW
ANI_MSG_CHANNEL_INFO_REQ	0x5	Channel info request message	To Host Driver
ANI_MSG_CHANNEL_INFO_RSP	0x6	Channel info response message	From Host Driver
ANI_MSG_OEM_ERROR	0x7	CLD-OEM error response	From FW

Message type ANI\_MSG\_OEM\_DATA\_REQ and ANI\_MSG\_OEM\_DATA\_RESP are used for different kinds of measurement requests/response. Some messages are passthrough messages for

Host driver if they are intended for firmware. The different subtypes of supported ANI\_MSG\_OEM\_DATA\_REQ and ANI\_MSG\_OEM\_DATA\_RESP messages are shown below.

Subtype ID	Value	Message ID where used
TARGET_OEM_CAPABILITY_REQ	0x1	ANI_MSG_OEM_DATA_REQ
TARGET_OEM_CAPABILITY_RSP	0x2	ANI_MSG_OEM_DATA_RSP
TARGET_OEM_MEASUREMENT_REQ	0x3	ANI_MSG_OEM_DATA_REQ
TARGET_OEM_MEASUREMENT_RSP	0x4	ANI_MSG_OEM_DATA_RSP
TARGET_OEM_ERROR_REPORT_RSP	0x5	ANI_MSG_OEM_ERROR
TARGET_OEM_CONFIGURE_LCR	0x9	ANI_MSG_OEM_DATA_REQ
TARGET_OEM_CONFIGURE_LCI	0xA	ANI_MSG_OEM_DATA_REQ
TARGET_OEM_CONFIGURE_WRU	0x80	ANI_MSG_OEM_DATA_REQ
TARGET_OEM_CONFIGURE_FTMRR	0x81	ANI_MSG_OEM_DATA_REQ

## 5.12 Support for dynamic LAN and WAN for single net device

Until QCA\_Networking\_2016.SP4.0 release, certain specific boards such as DK05 support single net device, and for the boards with single netdevices, adding VLANs and changing port-composition on run-time was not supported. Starting with QCA\_Networking\_2017.SP5.0 release, dynamic LAN and WAN for single net devices are supported. With this feature, even the boards with single netdevices can change their network configuration at run-time with exported sysctls and change the port-constitution for VLANs.

Previously, a limitation was present in the existing framework wherein Group0 always needs to have 1 port and link-detection will be enabled for this group, while Group1 always needs to have multiple ports with port-link detection always disabled for Group1. With the new implementation, even Group0 can have multiple ports and link-detection is always subject to number of ports in a specific group, which implies that with just one port in Group1, link-detection can be enabled for Group1.

Also, a limitation was present where-in any Group1, with single port, always had Link detection enabled by default. With the new implementation user can disable this polling of the link by essedma, and thereby disable link-detection completely for all the groups. This capability is called dynamic polling, and this can be configured with dts entry '**poll\_required\_dynamic**'. When **poll\_required\_dynamic** is set to '0' link detection is disabled for the specified group.

### 5.12.1 LAN/WAN group compositions

This section describes the different combinations of LAN and WAN groups that are supported.

#### Default 2 VLANs (1 Group 1 port + 4 Group 2 port)

Configure the DTSI configuration file, qcom-ipq40xx.dtsi, as follows:

```
aliases {
    spi0 = &spi_0;
    spi1 = &spi_1;
```

```

        i2c0 = &i2c_0;
        i2c1 = &i2c_1;
        ethernet0 = "/soc/edma/gmac0";
        ethernet1 = "/soc/edma/gmac1";
    };
edma@c080000 {
    .
    .
    qcom,rx_head_buf_size = <1540>;
    qcom,num_gmac = <2>;
    gmac0 {
        local-mac-address = [000000000000];
        qcom,poll_required = <0x1>;
        qcom,poll_required_dynamic = <1>;
        qcom,phy_mdio_addr = <4>;
        qcom,forced_speed = <1000>;
        qcom,forced_duplex = <1>;
        vlan_tag = <1 0x20>;
    };
    gmac1 {
        local-mac-address = [000000000000];
        qcom,poll_required_dynamic = <1>;
        vlan_tag = <2 0x1e>;
    };
}

```

Configure Linux network settings in the file at /etc/config/network:

```

config switch
    option name 'switch0'
    option reset '1'
    option enable_vlan '1'

config switch_vlan
    option device 'switch0'
    option vlan '1'
    option ports '0t 1 2 3 4'
config switch_vlan
    option device 'switch0'
    option vlan '2'
    option ports '0t 5'

```

### 5.12.1.1 2 VLANs with 1 Group 1 port + 4 Group 2 ports

A user can perform the changes in the default configuration and add a different port to Group1 using the sysctls as shown in this section. Here port 3 is added to Group1 instead of Port 5.

Sysctl entries can be decoded as follows, where 6 bit represent bitmask for each port with Bit0 representing CPU Port (which will always be 0), Bit1 representing Port1, Bit2 is port2 and so on.

	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
Group1	0	0	1	0	0	0	is equal to 001000 = 0x8
Group1	1	1	0	1	1	0	is equal to 001000 = 0x36

Configure the sysctl as follows:

```
cd /proc/sys/net/edma
echo 0x8 > default_group1_bmp
echo 0x36 > default_group2_bmp
echo 0 > default_group1_vlan_tag
echo 0 > default_group2_vlan_tag
```

In this configuration, all ports will be part of the netdevice corresponding to Group1, that is, eth0. Because Group1 has more than 1 port, link detection will be disabled for Group1

Configure Linux network settings in the file at /etc/config/network:

```
config switch
    option name 'switch0'
    option reset '1'
    option enable_vlan '1'

config switch_vlan
    option device 'switch0'
    option vlan '1'
    option ports '0t 1 2 4 5'
config switch_vlan
    option device 'switch0'
    option vlan '2'
    option ports '0t 3'
```

### 5.12.1.2 0 VLANs with 5 Group 1 ports + 0 Group 2 ports

Configure the sysctl as follows:

```
cd /proc/sys/net/edma
echo 0x3e > default_group1_bmp
echo 0x0 > default_group2_bmp
echo 0 > default_group1_vlan_tag
echo 0 > default_group2_vlan_tag
```

In this configuration, all ports will be part of the netdevice corresponding to Group1, that is, eth0. Because Group1 has more than 1 ports, link detection will be disabled for Group1

Configure Linux network settings in the file at /etc/config/network:

```
config interface 'lan'
    option ifname 'eth0'
    option force_link '1'
    option type 'bridge'
    option proto 'static'
    option netmask '255.255.255.0'
                option ipaddr '192.168.1.9'

config switch
    option name 'switch0'
    option reset '1'
    option enable_vlan '0'

config switch_vlan
    option device 'switch0'
```

```
option vlan '0'
option ports '0 1 2 3 4 5'
```

### 5.12.1.3 0 VLANs with 0 Group 1 ports + 5 Group 2 port

Configure the sysctl as follows:

```
cd /proc/sys/net/edma
echo 0x0 > default_group1_bmp
echo 0x3e > default_group2_bmp
echo 0 > default_group1_vlan_tag
echo 0 > default_group2_vlan_tag
```

In this configuration, all ports will be part of the netdevice corresponding to Group2, that is, ***eth1***. Because Group2 has more than 1 ports, link detection will be disabled for Group2

Configure Linux network settings in the file at /etc/config/network:

```
config interface 'lan'
  option ifname 'eth1'
  option force_link '1'
  option type 'bridge'
  option proto 'static'
  option netmask '255.255.255.0'
    option ipaddr '192.168.1.9'

config switch
  option name 'switch0'
  option reset '1'
  option enable_vlan '0'

config switch_vlan
  option device 'switch0'
  option vlan '0'
  option ports '0 1 2 3 4 5'
```

### 5.12.1.4 2 VLANs with 4 Group 1 ports + 1 Group 2 port

Configure the sysctl as follows:

```
cd /proc/sys/net/edma
echo 0x1e > default_group1_bmp
echo 0x20 > default_group2_bmp
echo 2 > default_group1_vlan_tag
echo 1 > default_group2_vlan_tag
```

In this configuration, ports 1,2,3,4 are part of Ethernet interface corresponding to Group1(eth0) and Port5 corresponds to Group2(eth1). Because Group2 has only 1port, link detection will be enabled for Group2(eth1) and as Group1 has more than 1 ports, link detection will be disabled for Group1(eth0).

This is a sample configuration which adds Ports 1,2,3,4 to Group 1. This can also be changed for instance. Port 1,2,4,5 can be part of Group1 and Port 3 can be part of Group2. Any combinations of ports can be added to any group.

Configure Linux network settings in the file at /etc/config/network:

```
config switch
    option name 'switch0'
    option reset '1'
    option enable_vlan '1'

config switch_vlan
    option device 'switch0'
    option vlan '1'
    option ports '0t 5'
config switch_vlan
    option device 'switch0'
    option vlan '2'
    option ports '0t 1 2 3 4'
```

After system bootup, user can switch from any of the previously mentioned combination to the other, by executing the sysctls and making corresponding changes in /etc/config/network and restarting network with /etc/init.d/networking restart.

## 5.13 TDMA method

Time division multiple access (TDMA) is a channel access methodology for point-to-multi-points within intra-backhaul network. TDMA deployment is controlled and managed by OEMs as a proprietary ecosystem and no interoperability is necessary with other Wi-Fi based TDMA devices.

TDMA is beneficial in reducing interferences between multiple APs operating within intra-backhaul network. STAs connected to AP start data transfer only on the reception of trigger frame from AP. AP coordinates and sends trigger frames to connected STAs in round robin manner.

This feature enables the user to configure TDMA support in the AP/STA at runtime. After TDMA is configured, AP sends trigger frames to STAs to indicate STA to start transmission. STA sends a burst of traffic and sends end-trigger to AP.

AP coordinates connected clients Best effort (BE) traffic by sending trigger frames to each of the clients in round robin scenario. STA starts transmitting only after receiving the trigger; therefore, collisions are avoided in the medium since only one STA can start transmission at any given time.

The design involves triggers being sent from both AP and STA.

- Trigger to STA is BAR frame.
- Trigger to AP from STA is PSPOLL frame.

A configuration command is available to turn on and turn off TDMA support at both STA and AP. Issue the `iwpriv wifiX qboost_enable 1` command in all the STA and AP boards to enable TDMA functionality. Issue the `iwpriv wifiX qboost_enable 0` command in all the STA and AP boards to disable TDMA functionality.

### 5.13.1 Sample configuration scenario

Run ping traffic on a single client after enabling TDMA support in AP and STA connected using the `iwpriv wifiX qboost_enable 1` command.

Consider the following AP configuration:

```
config wifi-device wifi0
    option type      qcawifi
    option channel   auto
    option macaddr  00:a0:c6:01:21:38
    option hwmode    11ng
    # REMOVE THIS LINE TO ENABLE WIFI:
    option disabled 1

config wifi-iface
    option device    wifi0
    option network   lan
    option mode      ap
    option ssid      Openwrt
    option encryption none

config wifi-device wifi1
    option type      qcawifi
    option channel   36
    option macaddr  00:a0:c6:01:20:da
    option hwmode    11ac
    # REMOVE THIS LINE TO ENABLE WIFI:
    option disabled 0

config wifi-iface
    option device    wifi1
    option network   lan
    option mode      ap
    option ssid      ROOT
    option encryption none
```

Consider the following STA configuration:

```
config wifi-device wifi0
    option type      qcawifi
    option channel   auto
    option macaddr  00:a0:c6:01:21:38
    option hwmode    11ng
    # REMOVE THIS LINE TO ENABLE WIFI:
    option disabled 1

config wifi-iface
    option device    wifi0
    option network   lan
    option mode      sta
    option ssid      Openwrt
    option encryption none
    option wds 1

config wifi-device wifi1
    option type      qcawifi
    option channel   auto
    option macaddr  00:a0:c6:01:20:da
    option hwmode    11ac
    # REMOVE THIS LINE TO ENABLE WIFI:
```

```

        option disabled 0

config wifi-iface
    option device    wifi1
    option network   lan
    option mode      sta
    option ssid      ROOT
    option encryption none
    option wds 1

```

Issue the `iwpriv wifiX qboost_enable 1` command in all the STA and AP boards to enable TDMA functionality. Keep the setup idle and check statistics 18 in STA and 16 in AP. Start traffic for all the clients and check stats 18 in STA connected. Verify using the sniffer that clients transmit unicast Best Effort QOS data only after reception of BAR trigger frame from AP.

Issue the `iwpriv wifiX qboost_enable 0` in both STA and AP to disable TDMA functionality. Verify the statistics and sniffer that BAR triggers are not sent anymore.

Stats 18 indicates in STA the SIFS response data frames and triggers received. Stats 16 in AP side represents SU-BAR trigger transmissions to the clients connected.

The following is an example of firmware stats 16 in AP side:

```

root@OpenWrt:/# iwpriv ath1 txrx_fw_stats 16
[ 7992.290000] TX_SELFGEN Info:
[ 7992.290000]
[ 7992.300000] su_ndpa      :      0
[ 7992.300000] su_ndp       :      0
[ 7992.300000] su_bar       : 672941
[ 7992.310000] su_cts2self  :      29
[ 7992.310000] su_ndpa_err  :      0
[ 7992.310000] su_ndp_err   :      0
[ 7992.320000] mu_ndpa     :      0
[ 7992.320000] mu_ndp      :      0
[ 7992.320000] mu_brpoll_1  :      0
[ 7992.330000] mu_brpoll_2  :      0
[ 7992.330000] mu_bar_1    :      0
[ 7992.330000] mu_bar_2    :      0
[ 7992.340000] mu_cts2self  :      0
[ 7992.340000] mu_ndpa_err  :      0
[ 7992.340000] mu_ndp_err   :      0
[ 7992.350000] mu_brp1_err  :      0
[ 7992.350000] mu_brp2_err  :      0
[ 7992.360000]

```

The following is an example of firmware Stats 18 in STA side:

```

root@OpenWrt:/# iwpriv ath1 txrx_fw_stats 18
[ 8644.660000] SIFS RESP RX stats:
[ 8644.660000]
[ 8644.670000] ps-poll trigger      :      0
[ 8644.670000] uapsd trigger       :      0
[ 8644.680000] qboost trigger data[exp] :      0
[ 8644.680000] qboost trigger bar[exp]  : 667448
[ 8644.690000] qboost trigger data[imp]  :      0

```

```

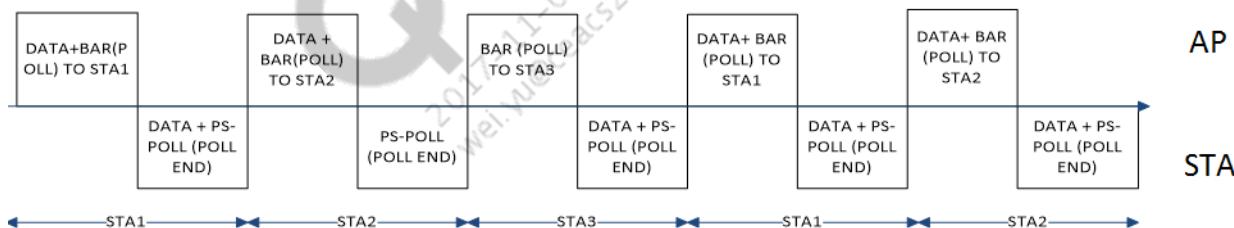
[ 8644.690000] qboost trigger bar[imp]   : 0
[ 8644.690000]
[ 8644.690000]
[ 8644.700000] SIFS RESP TX stats:
[ 8644.700000]
[ 8644.700000]   SIFS response data      : 1175
[ 8644.710000]   SIFS response timing err  : 382
[ 8644.710000]
[27618.444536] Ack RSSI: 27

```

## 5.13.2 Limitations

- MU is disabled.
- TxBF is disabled in STA side.
- Legacy mode 11a/11g/11b is not supported.
- Multiple VAPs in a single PHY radio are not supported. Only one VAP per radio is supported.
- Peak numbers and throughput to clients in multiclient mode are affected.

## 5.13.3 Sequence of frames exchange



### Frame Exchange Sequence

1. AP sends data to the STA followed by trigger frame and waits for STA to respond with data/end trigger.
2. After a STA receives a trigger, the STA can transmit data to the AP and end with PSPOLL trigger frame.
3. On reception of PSPOLL, AP schedules transmission of next STA in round robin manner.
4. If AP has no data to send to the STA, it sends trigger frame alone.
5. Upon reception of trigger frame from AP, if STA has no data to send to the AP, it sends trigger end frame alone.
6. If AP does not receive any data/end trigger from a STA. AP moves to the next STA after the timeout occurs.

## Scenario 1

Both AP and STA1 have data to transmit.

1. AP sends data to client along with BAR trigger frame to client.
2. AP starts a timer and waits for end trigger from STA1.
3. AP stops transmitting data and further BAR frames to STA1. STA1 alone has access to medium.
4. STA receives BAR trigger frame and sends traffic to AP for response timeout period.
5. Upon response timeout, STA sends the PSPOLL trigger frame to AP indicating end of traffic.  
STA stops transmitting further frames to AP.
6. AP starts servicing STA2.

## Scenario 2

AP has no data and STA2 has data to transmit.

1. AP sends BAR trigger frame to STA2.
2. AP starts a timer and waits for end trigger from STA2.
3. AP stops transmitting data frames and subsequent BAR frames to STA2. Only STA2 has access to the medium.
4. STA2 receives BAR trigger frame and sends traffic to AP for response timeout period.
5. Upon response timeout STA2 sends PSPOLL trigger frame to AP indicating end of traffic.  
STA2 stops transmitting further frames to AP.
6. AP starts servicing STA3.

## Scenario 3

AP has data to STA3 and STA3 has no data to transmit.

1. AP sends data to STA3 along with BAR trigger frame.
2. AP starts a timer and waits for end trigger from STA3.
3. AP stops transmitting data and subsequent BAR frames to STA3. Only STA3 has access to the medium.
4. STA3 receives BAR trigger frame and sends PSPOLL frame to AP to indicate the end of traffic.
5. AP starts servicing STA4.

## Scenario 4

Both AP and STA4 have no data to transmit.

1. AP sends BAR trigger frame to STA4.
2. AP starts a timer and waits for end trigger from STA4.
3. AP stops transmitting further BAR frames to STA4. STA4 alone has access to medium.

4. STA4 receives BAR trigger frame and sends PSPOLL frame to AP indicating end of traffic.
5. AP starts servicing STA5.

### Management Frame Exchanges

Management frames are queued all the time and will get opportunity to transmit at the end of any STA transmission. After every burst of traffic, management frames of AP and connected clients contend for the medium and get access to medium.

### Connection Frame Exchanges

EAPOL frames are queued with voice-access category. Therefore, they contend for the medium always and receive access for transmission.

### Multicast/ Broadcast Data

Multicast and broadcast data from AP will be sent via Voice access category and they contend for the medium always and get access for transmission.

## 5.14 Multi-credentials for VAP

When an AP is configured in security mode on two VAPs, the first VAP is always enabled with WPS. A functionality is introduced to enable STA to receive credential of both VAPs when STA connects with AP through WPS with first VAP. When one of the VAPs is down, the STA is able to connect to other VAP. This behavior is referred to as multi-credential of VAP.

Hostapd has option of extra\_cred to configure extra credential to be transmitted in M8 message of WPS. The extra\_cred parameter contains security configuration in binary format. AP always acts as a registrar to ensure STA does not change the configuration of AP. To enable multi-credential for VAP0, enter the following UCI command:

```
uci set wireless.@wifi-iface[0].multi_cred=1
```

With an AP in WDS mode, connect station to AP through WPS method to VAP0. The wpa\_supplicant config file is examined to verify whether both the VAP configurations are received and saved. If one of the VAPs in AP is brought down, the STA is able to connect to the other VAP.

The configuration settings of both the VAPs are passed to only to the client. AP is always in registrar mode. The order of VAPs is in the sequence in which the first two VAPs are connected. No random two VAPs are supported in a scenario that involves more than two VAPs. All rules of basic WPS continue to apply to both the VAPs.

# 6 Acceleration and Offload Features

---

This chapter describes the networking services (NSS) acceleration, and offload functionalities, such as TCP Segmentation Offload, Large Receive offload, checksum offload, Offload Optimized Host Data Path, NSS offload, HNAT Wi-Fi offload, and customization of full-offload models for peak performance.

## 6.1 TCP Segmentation Offload

The TCP Segmentation Offloading (TSO) consists of processing an extra large transmit frame (whose size is much larger than the WLAN PHY MTU), within the upper layers of the protocol stack. The extra large frame is divided into a series of segments that are smaller than the MTU for transmission over the WLAN PHY. Even if the TSO segmentation is done within the (host) CPU rather than the hardware, a CPU utilization benefit can be achieved by invoking the upper layers of the protocol stack less frequently than with the MTU-sized frames.

### 6.1.1 Implementation

Each packet transmitted is checked if it is a TSO packet or a plain TCP packet with the “qdf\_nbuf\_is\_tso” API. If it is a TSO packet, the “ol\_tx\_tso\_sg\_process\_skb” API is called for further processing; the actual segmentation of the packet is done by this API.

The “ol\_tx\_tso\_sg\_process\_skb” API initially allocates a software TSO descriptor and fills up the descriptor with information specific to the TSO packet. This descriptor is freed only when the last segment of the TSO packet is sent out. Next, the API iterates through the total payload length of the packet and fills the TSO software descriptor with the address of each segment and the segment length. Once the segment information is available with the TSO software descriptor, the “ol\_tx\_ll\_fast” API is called for further processing. The reference count for the TSO packet is also updated as this packet should not be freed until the last segment is transmitted.

The “ol\_tx\_ll\_fast” API prepares the “htt\_tx\_desc” from the TSO software descriptor by calling the “htt\_tx\_desc\_frag” and “ol\_tx\_tso\_sg\_desc\_prepare” APIs. Once the htt\_tx\_desc is prepared, the TSO packet is handed over to the copy engine for transmission.

### 6.1.2 Configuration

The ethtool utility is used to enable or disable the TSO feature.

```
ethtool -K ath0 tx on/off  
ethtool -K ath0 sg on/off  
ethtool -K ath0 tso on/off
```

The iwpriv tool is used to get and reset the TSO statistics.

```
iwpriv ath0 get_tso_stats
iwpriv ath0 rst_tso_stats
```

## 6.2 Large Receive Offload

In computer networking, Large Receive Offload (LRO) is a technique for increasing the inbound throughput of high-bandwidth network connections, by reducing the CPU overhead. It works by aggregating multiple incoming TCP packets from a single stream into a larger buffer before they are passed higher up the networking stack; this reduces the number of packets that have to be processed.

### 6.2.1 Implementation

The hardware MAC's LRO block provides information on whether the received packet is LRO eligible and the payload checksum value. This information is copied from the rx msdu descriptor to the skb cb array for further processing by the “htt\_set\_lro\_info\_ll” API.

If the LRO feature is enabled on the VAP, the received packet is processed by “qca\_lro\_receive\_skb” instead of the default “netif\_rx” API. The “qca\_lro\_receive\_skb” gets a LRO descriptor from the descriptor array. Based on the four tuple fields (src IP, dest IP, src port, and dest port) either an active LRO descriptor or a new LRO descriptor is returned from the pool.

An active descriptor is one that has already been initialized by a previous packet that matches the four tuples of the current packet (same stream). If this is a new LRO descriptor, it is initialized and the current packet will be the parent packet. Packets received thereafter with the same tuple info are appended to the parent packet by the “qca\_lro\_add\_packet” API. The aggregation is done until an out of sequence packet or a non-LRO packet is received, or if the MAX aggregation limit is reached. The packets are flushed to the stack using the “qca\_lro\_flush” API.

### 6.2.2 Configuration

The ethtool utility is used to enable or disable the LRO feature.

```
ethtool -K ath0 lro on/off
```

The iwpriv tool is used to get and reset LRO statistics.

```
iwpriv ath0 get_lro_stats
iwpriv ath0 rst_lro_stats
```

## 6.3 Checksum Offload

The AR900B hardware MAC provides L4 checksum calculation and verification in both the Rx and Tx data path. This helps in improving the network stack performance as this is offloaded to the hardware.

## 6.3.1 Implementation

### 6.3.1.1 Tx checksum offload

The network stack checks the device features (dev\_features) and calculates the Tx packet checksum. If the device feature advertises checksum offload capability, the IP\_SUMMED field of each Tx packet is set to “CHECKSUM\_PARTIAL”; this indicates that the device is required to checksum the packet, or set “CHECKSUM\_NONE” indicating that the packet is already checksummed by the protocol.

If ip\_summed is set to CHECKSUM\_PARTIAL, bits 16 to 20 of the third word of htt\_frag\_desc is set, informing the hardware to perform checksum calculation on the transmit packet.

```
cksum_enable = (uint32_t *) (((uint32_t *)tx_desc->htt_frag_desc) + 3);
*cksum_enable |= HTT_ENDIAN_BYTE_IDX_SWAP(0x1f0000);
```

**NOTE** This feature is disabled by default and the ethtool utility is used to enable it. The command to enable is:

```
ethtool -K ath0 tx on
```

### 6.3.1.2 Rx checksum offload

In the Rx path, the AR900B hardware MAC has the capability to verify the Rx packet checksum. The attention field of the Rx descriptor of the msdu indicates if the checksum is correct or incorrect. In case of a successful hardware verification, the ip\_summed field of the packet is set to “CHECKSUM\_UNNECESSARY” indicating the network protocol stack to ignore further checksum verification. However, if the hardware indicates checksum failure, the ip\_summed field of the packet is set to “CHECKSUM\_NONE” so that the network protocol stack should verify the received packet checksum.

The “htt\_set\_checksum\_result\_ll” API is used for Rx checksum offload.

**NOTE** This feature is enabled by default at the compile time via the “RX\_CHECKSUM\_OFFLOAD” macro; hence no further user configuration is required.

## 6.4 Offload Optimized Host Data Path

The offload data path has been optimized in multiple phases. All the major optimizations are kept under different compile time macros, which can be enabled or disabled. The macros provide a way for handling the data paths. All the macros are enabled by default.

- QCA\_OI\_11AC\_FAST\_PATH
- QCA\_OI\_TX\_CACHEDHDR
- QCA\_OI\_RX\_BATCHMODE
- QCA\_OI\_TX\_PDEV\_LOCK

The ATH\_11AC\_TXCOMPACT macro used for initial optimization has been overwritten with QCA\_OL\_11AC\_FAST\_PATH. The other three macros are active. The QCA\_OL\_11AC\_FAST\_PATH and QCA\_OL\_TX\_CACHEDHDR macros are used in the downlink path and QCA\_OL\_RX\_BATCHMODE is used in uplink. The QCA\_OL\_11AC\_FAST\_PATH macro is used for exception paths like TSO, multicast enhancement, scatter gather and QCA\_OL\_TX\_CACHEDHDR is used for standard data paths. These functionalities are separated by the top level functions, ol\_tx\_ll\_fast and ol\_tx\_ll\_cachedhdr.

The major difference between the two paths is the way the headers are filled and the number of CE descriptors used. In case of the ol\_tx\_ll\_fast path, a separate consistent memory is used to fill the host target transmit header, and sent as one of the fragments in addition to the skb data payload.

In case of the ol\_tx\_ll\_cachehdr path, the host target transport header is copied onto the skb payload and sent as a single fragment. In addition, a prefilled header data is used which is copied onto the skb payload, instead of modifying each bit of the header. This has many advantages like, access of skb payload which is a cached memory compared to a non-cached, consistent, prefilled header, and reduces the number of copy engine descriptors which halves the non-cached descriptor access.

The QCA\_OL\_TX\_PDEV\_LOCK macro has multiple locks taken at different stages like, vap transmit, and sw desc queue. The copy engine send is combined with a single global transmit lock.

The QCA\_OL\_RX\_BATCHMODE macro is a dynamic software rate based interrupt mitigation for incoming packets, coming from the target to host. The design for batch mode is implemented in the firmware. On the host, the macro is used for enabling or disabling the feature.

[Figure 6-1](#) shows the offload optimized transmit flow.

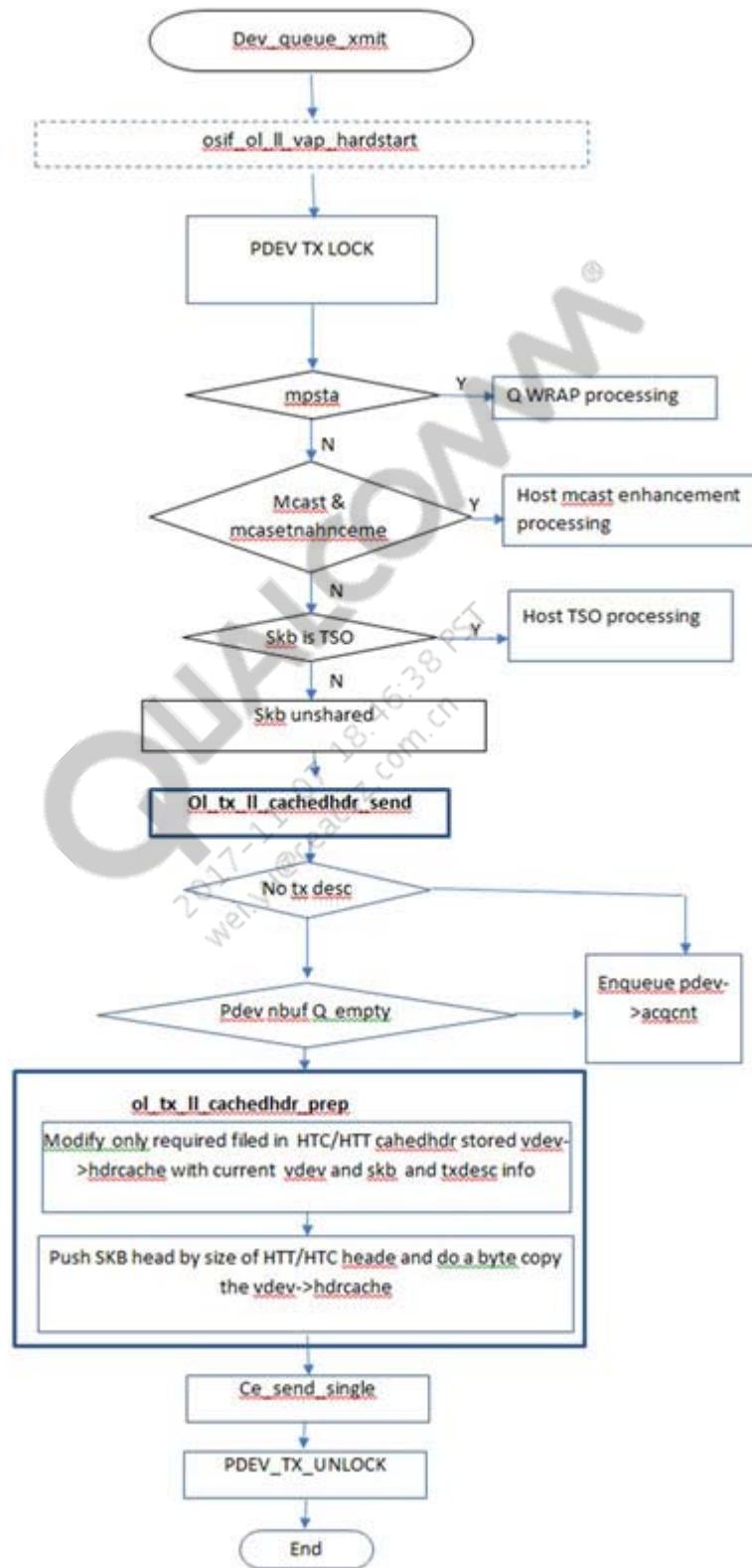


Figure 6-1 Offload Optimized Transmit Flow Diagram

Figure 6-2 shows the transmit completion and scheduling flow.

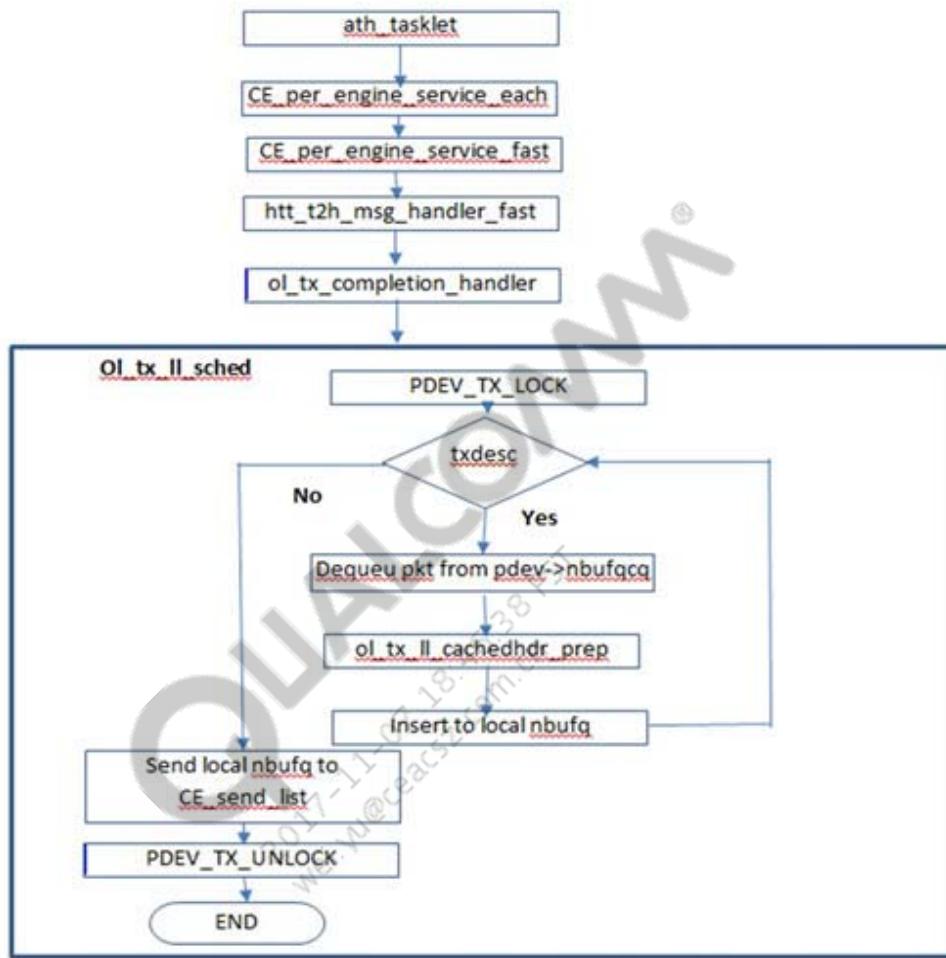


Figure 6-2 Transmit Completion and Scheduling Flow Diagram

## 6.5 NSS Wi-Fi Offload

The NSS Wi-Fi offload feature offloads host transmit and receive data plane to NSS ,UBI32 process, which helps to reduce the CPU load on IPQ806x host to a great extent.

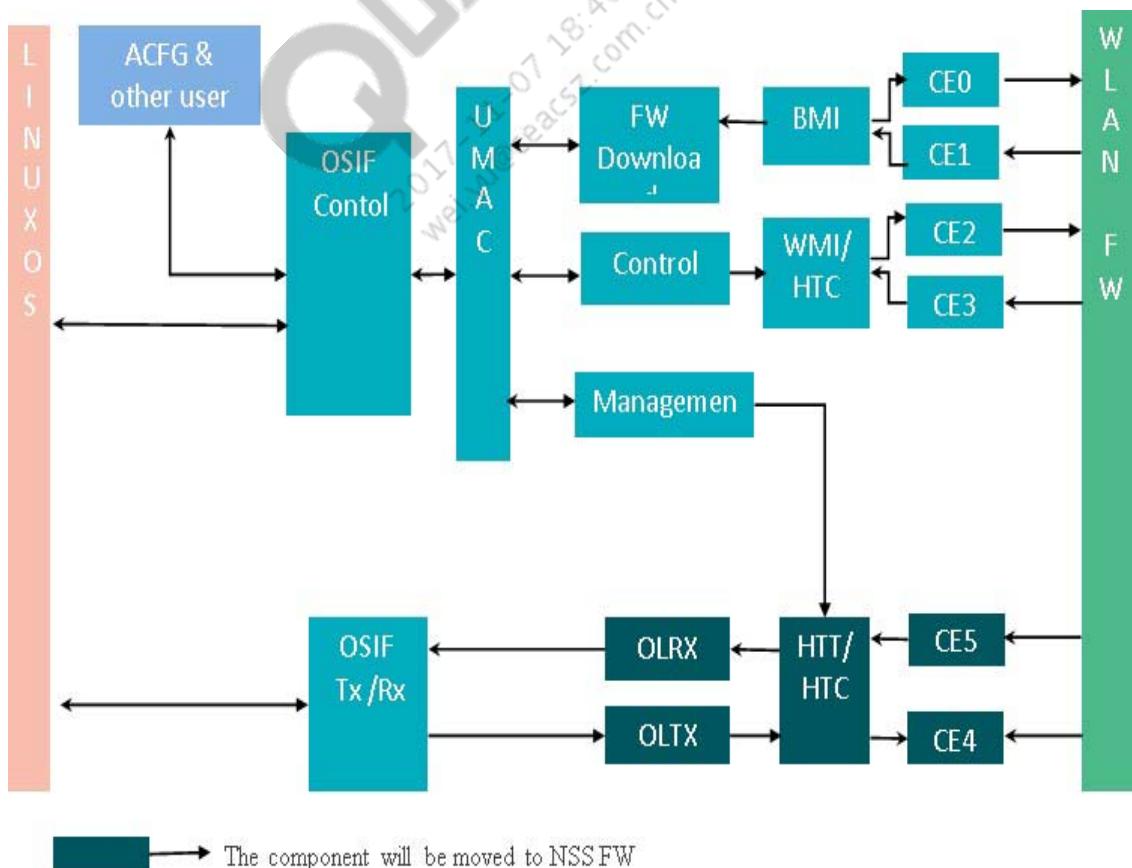
The feature is specific for IPQ806x platform.

The Wi-Fi Data Tx/Rx component moved to NSS and Selected copy\_engine (CE4/CE5) which are tied to the data path. The other component of wifi Beacon /management packets still go through normal Wi-Fi host driver. Through WMI interface and PCI driver attach, firmware download remains on the host.

### 6.5.1 Features

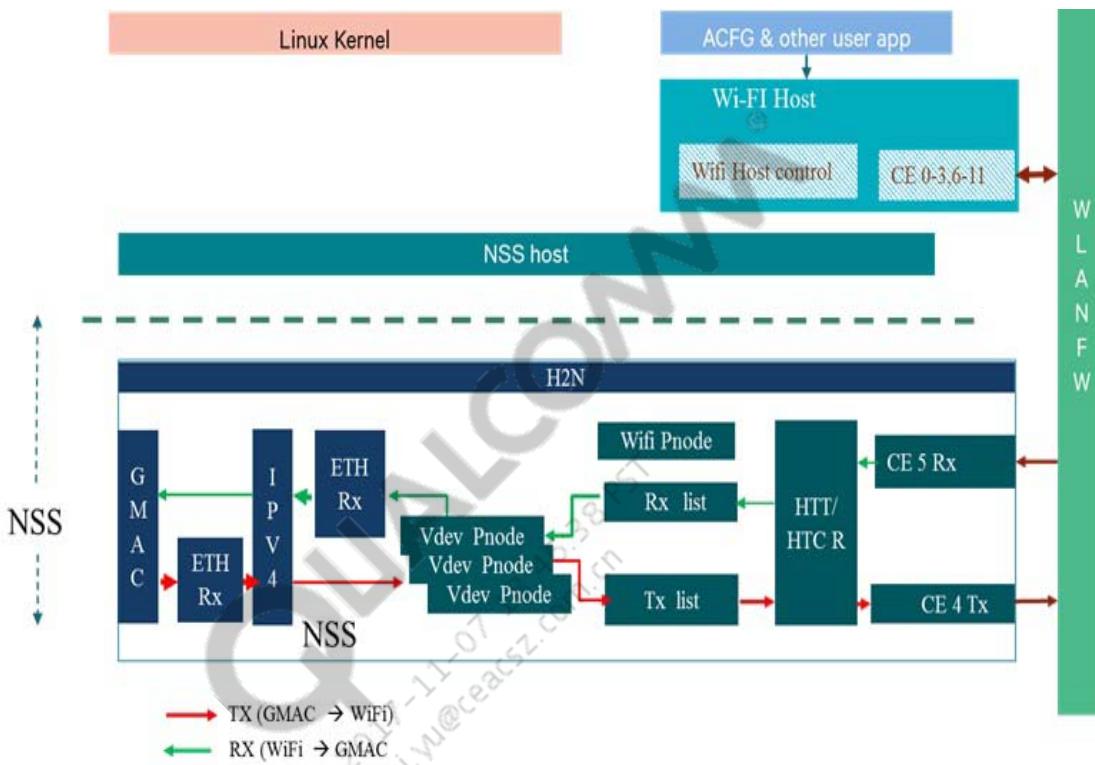
- Wi-Fi offload supports two mode as per NSS acceleration is enabled /disabled, these are controlled through NSS configuration on IPQ806x
  - Standard Profile – NSS acceleration is enabled
  - Enterprise Profile – NSS acceleration is disabled
- Common UMAC and user configuration between generic Wi-Fi release and nss Wi-Fi offload mode. This will provide seamless up gradation to NSS Wi-Fi offload mode for customer
- No change in Wi-Fi user configuration required.
- No changes in Wi-Fi firmware - This allows common Wi-Fi firmware binary to be used seamless across nss wifi offload and stand Wi-Fi release driver
- Commons Wi-Fi host driver binary. Wi-Fi offload mode is chosen on module load time.
- Run time co-existence with NSS redirect mode and standard Wi-Fi mode.e.g one of the radio can be kept in NSS Wi-Fi offload mode and other can be in NSS redirect mode.

### 6.5.2 Host Driver Split Architecture



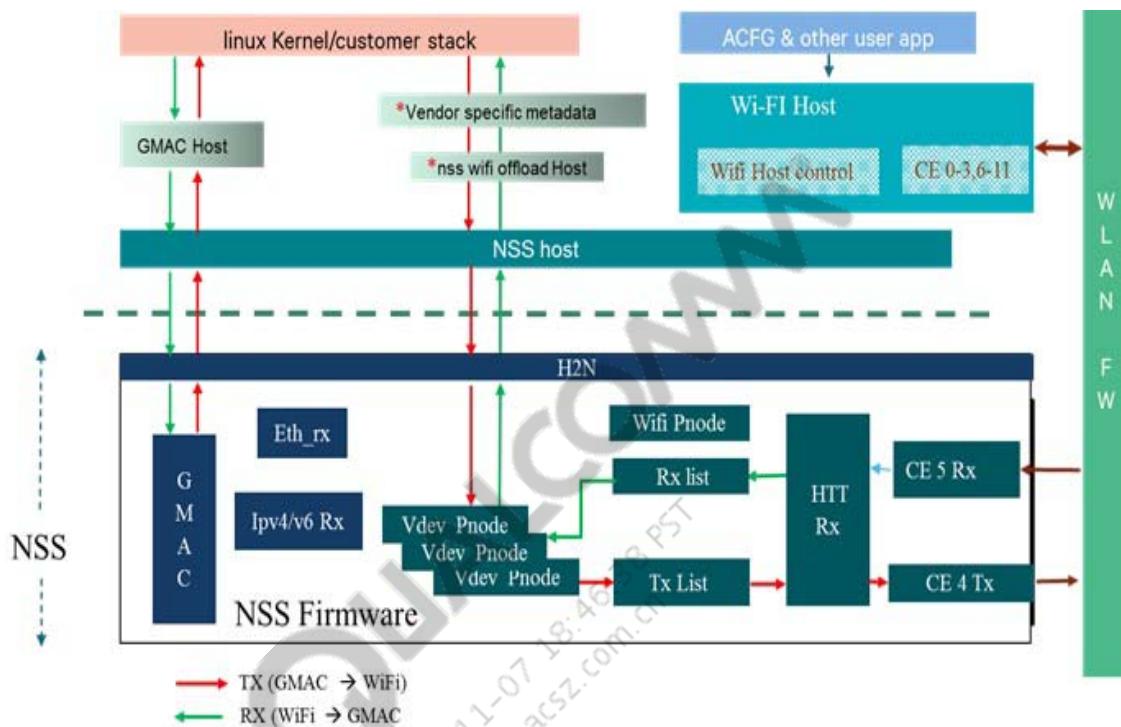
### 6.5.3 Standard Profile

Standard profile NSS fast path acceleration is enabled where the bringing and routing is done by NSS subsystem. In this mode none of the data packets traverse in host and hence very negligible CPU load is seen on host CPU.



## 6.5.4 Enterprise Profile

In case of Enterprise profile the NSS acceleration is disabled. The packets are send to host after Wi-Fi processing for bridging /routing.



## 6.6 HNAT Wi-Fi Offload

This section discusses the HNAT (Host Network Address Translation) Wi-Fi Offload feature.

### 6.6.1 QCA8327

The QCA8327 is a gigaport switch, integrated externally with the Qualcomm Technologies QCA955x RGMII/SGMII. The GMAC0 of the QCA955x is configurable whether SGMII or RGMII and GMAC1 in default with RGMII.

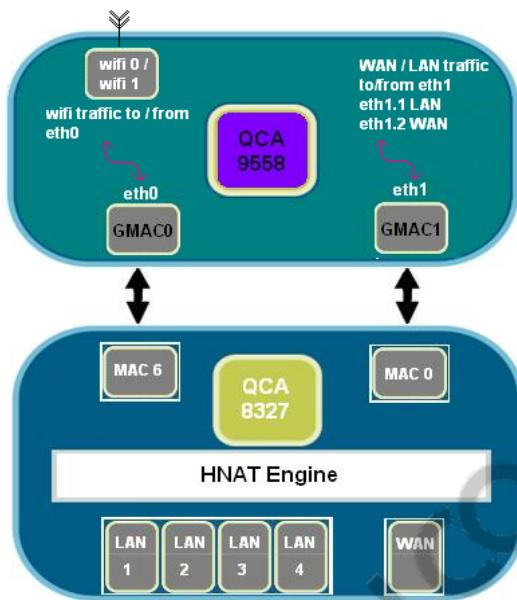
### 6.6.2 Hardware NAT Accelerator

#### 6.6.2.1 HNAT Engine

The QCA8327 gigaport switch supports both NAT and NAPT functionality. Its features include

- 128 ARP entry
- 1024 NAT entry
- 8 Router MAC address/VID table

- 16 Public IP table



**Figure 6-3 HNAT Block Diagram**

QCA9558 GMAC1 is used for LAN and WAN, separated by VLAN IDs, and GMAC0 is used for dedicated Wi-Fi traffic that pass thru between LAN and WAN interfaces. All Wi-Fi packets will be sent through MAC6 of QCA8327 into the LAN or WAN, or vice versa through a bridge. This bridge is used to only pass the traffic from MAC6 (LAN/WAN) to Wi-Fi and vice versa, does not have any IP assigned to it, and also does not respond to any ARP requests.

For HNAT engine learning, initially the packet will go through the software NAT module. After connection is established and the traffic reached the threshold value, the NAPT entry will be added into HNAT Engine. Subsequent packets are NATed by the HNAT engine, which forwards it to the destination port directly, without sending them to the QCA955x CPU. Non-NATed Wi-Fi traffic still pass thru the QCA955x CPU. The QCA955x CPU utilization will be reduced significantly once HNAT engine kicks in for NAT functionality.

### 6.6.2.2 HNAT: Packet Flow

#### Traffic path between WLAN to WAN

##### Software Path

Ath0→br0→eth0→s17-Mac6→s17-Mac0 →eth1→eth1.1→routing→eth1.2→eth1→s17-mac0→s17-mac5→WANPC

##### Hardware Path

Ath0→br0→eth0→s17-mac6→S17-HNAT→s17-mac5→WANPC

#### Traffic path between LAN to WAN

##### Software Path

LAN PC → s17-MAC [1-4] → s17-Mac0 → eth1 → eth1.1 → routing → eth1.2 → S/W  
 NAT → eth1 → s17-mac0 → s17-mac5 → WANPC

### **Hardware Path**

LANPC → s17-MAC [1-4] → S17HNAT → s17-mac5 → WANPC

In both cases, initially all the packets will go through the software path. Once the NAPT entry is added into the S17 HNAT table, the packets follow the hardware path.

## **6.6.3 Configuration**

The following commands should be executed on the AP to configure it for HNAT Wi-Fi Offload. The AP (WAN interface) should not be in bridged mode. Run the Ixia ixChariot console on the WAN PC (IP address 10.10.2.xxx) and Wi-Fi. The LAN PC IP should be 192.168.1.xxx.

- brctl delif br0 eth1
- ifconfig br0 0.0.0.0 up
- brctl addif br0 eth0
- vconfig add eth1 1
- vconfig add eth1 2
- ifconfig eth1.1 192.168.1.2 up
- ifconfig eth1.2 10.10.2.1 up
- ifconfig eth1.2 netmask 255.255.255.0
- echo 1 >/proc/sys/net/ipv4/ip\_forward
- **echo 1 > /proc/sys/net/ipv4/conf/br0/arp\_ignore**

### **IPTables examples**

- iptables -t nat -A POSTROUTING -o eth0.2 -j MASQUERADE  
 iptables -t nat -A PREROUTING -i eth0.2 -j DNAT --to 192.168.1.201

Bring up the AP and be sure to ping between all PCs. Do not add both eth0 and eth1 into bridge. This avoid a loop since both ports are in same VLAN.

## **6.7 Customize full-offload configuration for peak KPI**

This section describes the configuration specifications required on full-offload chipsets, namely QCA9531, QCA9558, and QCA9563 chipsets, to achieve peak performance in the full-offload architecture in QCA\_Networking\_2016.SPF4.0 CSU1 and later releases that are based on QSDK. These customizations are required because QSDK does not completely support the full-offload configuration. These configuration changes must be performed after the image is loaded from flash memory and the device is booted using the image from the flash.

## 6.7.1 Load the full-offload modules

The following is the list of modules and the parameters passed to them. Insert the target modules before inserting the host modules.

### 6.7.1.1 Host modules—Flash mode

After inserting the target modules in the target board, the existing Wi-Fi modules in the host must be unloaded and VLAN must be created before loading the full-offload modules. Enter the following commands:

```
$ wifi unload;
$ cd /lib/modules/3.3.8;
$ vconfig add eth1 2
$ ifconfig eth1.2 up;
$ brctl addbr br-lan
$ ifconfig br-lan 192.168.1.1
$ Ifconfig br-lan up
$ rmmod atd.ko
$ rmmod hif_gmac.ko
$ rmmod fwd.ko
$ rmmod adf.ko
$ insmod adf.ko
$ insmod hif_gmac.ko tgt_if=eth1.2 rom_mode=0
$ insmod mdio.ko
$ insmod fwd.ko
$ insmod atd.ko
```

### 6.7.1.2 Target modules – Flash mode

Create a VLAN before inserting the full offload modules as follows:

```
$ vconfig add eth1 2
$ ifconfig eth1.2 up
$ cd /lib/modules/3.3.8
$ insmod hif_gmac hst_if=eth1 rom_mode=0
$ insmod atd.ko
$ wifi load
```

### 6.7.1.3 Host modules – BootROM mode

The following commands will download the stage1 and stage2 images to the target and boot the target SOC:

```
$ cd /lib/modules/3.3.8
$ insmod adf.ko
$ insmod hif_gmac.ko tgt_if=eth0
$ insmod mdio.ko
$ insmod fwd.ko
$ insmod atd.ko
```

#### 6.7.1.4 Target modules – BootROM mode

After the target is booted, the target modules can be inserted with the following commands.

```
$ cd /lib/modules/3.3.8
$ insmod hif_gmac hst_if=eth0
$ insmod atd.ko
$ wifi load
```

#### 6.7.2 Modify the full-offload network configuration for flash mode

Network configuration is different for full-offload from the configuration for partial-offload because the host and the target are connected using GPRS MAC (GMAC). The default QSDK network configuration conflicts with this setting. Apply the following configuration on host and target modules so that they can communicate with each other. Change the bridge configurations on the host and target modules as necessary.

The following is the configuration on the host module:

```
config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
    option ipaddr '127.0.0.1'
    option netmask '255.0.0.0'

config interface 'lan'
    option ifname 'eth0 eth1'
    option type 'bridge'
    option proto 'static'
    option ipaddr '192.168.1.5'
    option netmask '255.255.255.0'

config interface 'wan'
    option ifname 'eth0'

config switch
    option name 'switch0'
    option reset '1'
    option enable_vlan '1'

config switch_vlan
    option device 'switch0'
    option vlan '2'
    option ports '0 1 2 3 4'

config switch_vlan
    option device 'switch0'
    option vlan '1'
    option ports '5 6'
```

The following is the configuration on the target module:

```
config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
```

```

        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'

    config interface 'lan'
        option ifname 'eth1'
        option type 'bridge'
        option proto 'static'
        option ipaddr '192.168.1.6'
        option netmask '255.255.255.0'

    config interface 'wan'
        option ifname 'eth0'

    config switch
        option name 'switch0'
        option reset '1'
        option enable_vlan '1'

    config switch_vlan
        option device 'switch0'
        option vlan '2'
        option ports '0 1 2 3 4'

    config switch_vlan
        option device 'switch0'
        option vlan '1'
        option ports '5 6'

```

### 6.7.3 Modify the Ethernet MTU configuration

Because the full-offload model adds additional headers to the packets that are sent across the GMAC link, the existing MTU must be increased. This increase is necessary in both the Ethernet switch and the Ethernet interface levels.

To change the Ethernet MTU settings on the host and target modules at the Ethernet switch level, insert the qca-ssdk.ko module if not previously inserted and then enter the command to set the frame size to 2048.

Enter the following commands on the host:

```

$ cd lib/modules/3.3.8; insmod qca-ssdk.ko; cd -
$ ssdk_sh misc framemaxSize set 2048
$ ssdk_sh misc framemaxSize get

```

Enter the following commands on the target:

```

$ swconfig dev switch0 set max_frame_size 2048
$ swconfig dev switch0 get max_frame_size. (The value should be >= 2048)

```

To change the Ethernet interface MTU settings on the host and target modules at the Ethernet interface level, enter the following commands:

```

$ ifconfig eth0 mtu 2048
$ ifconfig eth1 mtu 2048

```

## 6.7.4 Reduce CPU utilizations

QSDK is loaded with several applications, which consume CPU on a per-packet basis. These applications must be disabled to free up the CPU memory for full-offload performance.

To disable firewall, enter the following command on host:

```
$ /etc/init.d/firewall stop
```

To disable logging applications, enter the following command on host and target:

```
$ killall syslogd
$ killall klogd
```

## 6.7.5 Disable kernel prints

Kernel prints also are known to cause reductions in the throughput, especially due to the Ethernet flow control. The prints need to be disabled to see smoother graph. Enter the following command on host and target modules:

```
dmesg -n 1
```

## 6.7.6 Sigma DUT configurations

UCI and Wi-Fi commands were not supported during full-offload development because only ACFG was required. However, because Sigma-DUT is dependent on UCI and Wi-Fi commands, only basic VAP creation and deletion, SSID, and modes are supported. All security modes are supported using UCI and Wi-Fi commands.

## 6.7.7 Bypass-LAN configuration

In the bypass-LAN mode, the host is bypassed and all packets are forwarded to the Ethernet driver from the atd module of the target.

### Target commands

1. Bypass daemon:

After the host driver initializes the full-offload Radios in the target, the bypass-daemon, which configures the bridge and the interface need to be run on the Ethernet link to the Host. This is a one-time command, which is run during bootup.

```
# bypd <ethX> &
```

ethX – the interface on which the Host is connected.

### Host commands

1. Enable bypass mode:

After the host full-offload modules are initialized, bypass tunnel mode is enabled using the following commands. The commands creates the bridge in the target. This is a one-time command after the bypass daemon in the target is initialized.

```
# byp_cli enable tunnel
```

2. Enter the filter command:

After the VAPs are created, the following `byp_cli` is added to filter packets destined to the VAP and pass it to the Host from the Target.

```
# bypass_cli filteradd <VAP MAC>
```

3. Enter the bridge command:

After the filter command is issued, the VAP should be added in the Target's Bridge using the following command.

```
# bypass_cli vapadd < athx>
```

athX – VAP interface name.

### 6.7.8 Limitation

While booting up, the target and host must be in synchronization. Make sure that the target modules are inserted before the host modules.

## 6.8 IPQ806x IP Crypto Offload

IPQ806x provides a total of four Crypto Engines for accelerating any kind of cryptographic operations required by the Host software. These engines are capable of operating independently on discrete data using same or different crypto algorithms.

Some of the capabilities of the engine are listed below:

- Ability to store keys in a cached fashion for repeated crypto transactions; the maximum number of cached keys that can be provisioned from all the four crypto engines is 16.
- Ability to generate ICV (Hash) in memory or through registers; for Internet Protocol Security (IPsec) hash can be generated inside the data packet at the trailer section.

Refer to the IPQ806x hardware documentation for a complete feature set of the crypto engines:

- DB149-01x Hardware Reference Guide (80-Y7523-1)
- AP148-01x Hardware Reference Guide (80-Y7526-1)

### 6.8.1 Driver

The driver exposes all the four Crypto Engines as a single virtual crypto device to the Host software. It will perform the task of distributing crypto requests among these engines and collecting them back upon completion.

The crypto operations are identified using a session ID that must be allocated before initiating any crypto operation through the driver. It is the responsibility of the Host software to ensure that all crypto operations are initiated with a valid session ID.

Each session ID defines the context of a crypto operation and has the following elements associated with it:

- Cipher key and length; key used for encryption or decryption of data (e.g., 128, 256 etc.)

- Cipher algorithm; algorithm used for encryption or decryption of data (e.g., AES, 3DES etc.)
- Authentication key & length; key used for authenticating the data (e.g., 96, 128, 256 etc.)
- Authentication algorithm; algorithm used for authenticating the data (e.g., SHA1, SHA2 etc.)

A session can store the keys in a cached or un-cached mode depending upon the availability of cached slots in the hardware and may not necessarily map to the maximum available slots in HW (16 slots). The policy to allocate the cached slots is on a first come first serve basis and once all the available slots are used the keys will be stored in DDR memory. The advantage of using a cached key slot is in per-packet performance whereby the extra DDR fetch for the keys (cipher and authentication) are absent.

Each session will map to a flow running through the system, where a flow is assumed to be repeating crypto operations sharing same keys and algorithms on different data. The driver will use multiple crypto engines to increase the overall flow performance but maintain a strict ordering within it. Multiple sessions do not have ordering restriction across themselves which helps in improving the overall performance, for example, a session used for encryption path can simultaneously run, without any ordering restriction with a session used for decryption (classic TCP flow through an IPsec tunnel).

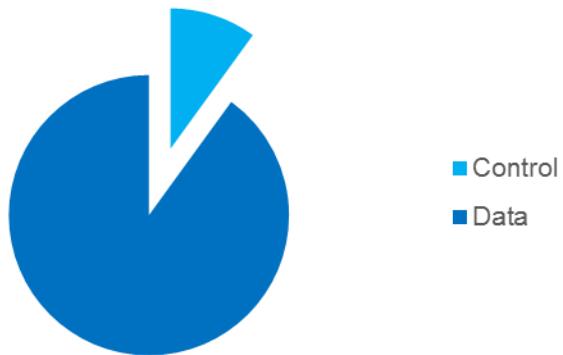
### 6.8.2 Offload

The crypto driver is essentially comprised of the following high level components:

- Control—Used for HW initialization and session specific programming such as keys, algorithms, and so on.
- Data—Used for executing the crypto operation like distributing across multiple engines, collecting completed operations from multiple engines, ordering the completed results and indicating it up to the host SW. Also, it needs to be at-least SMP safe so that it can be driven from multiple Host CPU cores.

The following figure describes the distribution of the crypto CPU load among its components if the driver runs both the components from the host processor.

**Crypto driver's CPU load distribution**



**Figure 6-4 Load distribution among components**

To reduce the amount of CPU load used in the data path, a significant set of operations were moved to one of the multi-threaded network accelerator engines cores. The remaining data path in Host performs only transportation of crypto requests from Host CPU to multi-threaded network accelerator engines core and back, the CPU loading cost of which is equivalent to the cost any non-crypto transaction, like GMAC packet receive/transmit using the NSS HLOS driver.

### 6.8.3 Crypto Offload APIs

The driver exposes a set of APIs for doing crypto transaction; these are completely agnostic to the higher level applications such as IPsec, SSL, etc., and provide a common gateway through which multiple applications can use the crypto HW simultaneously.

This section covers some of the key APIs that should be used to interact with the crypto driver; for the complete list of APIs and its description can be found in API documentation or in **nss\_crypto\_if.h** in the source code.

#### 6.8.3.1 User registration

Name	<code>nss_crypto_register_user</code>
Input	Attach and detach functions of the user application
Output	None

This API is used for registering multiple applications with the crypto driver. It will take inputs as the attach function called when the SW is ready to serve the application, and a detach function called when the application de-registers itself or the driver goes away.

The handle or cookie exchange happens at the time of attach function call where the crypto driver provides the user/application its handle and the user application in return gives its handle. These handles should be used for all future communications between the user and the driver.

#### 6.8.3.2 Buffer allocation

Name	<code>nss_crypto_buf_alloc</code>
Input	Crypto user handle
Output	Crypto buffer

A crypto buffer is an information bundle that describes a crypto transaction, which the HW must perform on the data/memory pointed by the crypto buffer. Various fields in the buffer structure are used to send this information down to the HW.

Each buffer contains fields for storing a cookie and a completion callback function. These are used to make the completion callback to the user when the buffer is marked completed by the driver. This provides the flexibility to the user to categorize its completion on a per crypto transaction level without paying the penalty of demultiplexing or classifying it after the transformation.

This API allocates the buffer from an internal pool, which is replenished whenever the user frees up the buffers. The user must handle the situation where all the buffers in the pool have been exhausted; which can happen either due to a large burst of requests submitted to the hardware or resource exhaustion at NSS side. In both the cases, it can either recover from the situation by itself or require an external trigger to come out.

### 6.8.3.3 Session allocation

Name	nss_crypto_session_alloc
Input	Crypto user handle, cipher key, authentication key
Output	Session index

A session allocation is required before requesting any crypto transaction to the HW. This prepares the SW and HW states for a new set of transactions sharing the same keys and algorithms. This call is synchronous in nature whereby a return value of the call will describe the success or failure.

It is the responsibility of the user to make sure that a crypto transaction has a valid session index before submitting requests to the driver.

### 6.8.3.4 Crypto transformation

Name	nss_crypto_transform_payload
Input	Crypto user handle, crypto buffer
Output	Status of the call

This API is used to submit a crypto request to the HW. The HW performs the transformation based upon the information sent in the crypto buffer and asynchronously indicates its completion.

The status of this call only indicates whether the request was successfully sent to NSS. In case of failure the user must have mechanisms to either retry the request or release it along with all allocated resources such as freeing up the crypto buffers.

## 6.8.4 Supported Crypto Algorithms

The following table lists the crypto cipher/authentication algorithms, HW and SW support, and their operation types.

Operation Type	Algorithm	HW Support	SW Support
AEAD	AES128_CBC_SHA1_HMAC	IPQ806x, IPQ807x	Yes
AEAD	AES256_CBC_SHA1_HMAC	IPQ806x, IPQ807x	Yes
AEAD	AES128_CBC_SHA256_HMAC	IPQ806x, IPQ807x	Yes

AEAD	AES256_CBC_SHA256_HMAC	IPQ806x, IPQ807x	Yes
AEAD	AES128_CTR_SHA1_HMAC	IPQ806x, IPQ807x	Yes
AEAD	AES256_CTR_SHA1_HMAC	IPQ806x, IPQ807x	Yes
AEAD	AES128_CTR_SHA256_HMAC	IPQ806x, IPQ807x	Yes
AEAD	AES128_CTR_SHA256_HMAC	IPQ806x, IPQ807x	Yes
AEAD	3DES_CBC_SHA1_HMAC	IPQ806x, IPQ807x	Yes
AEAD	3DES_CBC_SHA256_HMAC	IPQ806x, IPQ807x	Yes
AEAD	AES128_CCM	IPQ806x, IPQ807x	No
AEAD	AES128_GCM	IPQ807x	No
AEAD	AES128_CBC_MD5_HMAC	IPQ807x	No
AEAD	AES256_CBC_MD5_HMAC	IPQ807x	No
AEAD	AES128_CTR_MD5_HMAC	IPQ807x	No
AEAD	AES256_CTR_MD5_HMAC	IPQ807x	No
AEAD	3DES_CBC_MD5_HMAC	IPQ807x	No
ABLK	AES128_CBC	IPQ806x, IPQ807x	Yes
ABLK	AES256_CBC	IPQ806x, IPQ807x	Yes
ABLK	AES128_CTR	IPQ806x, IPQ807x	Yes
ABLK	AES256_CTR	IPQ806x, IPQ807x	Yes
ABLK	3DES_CBC	IPQ806x, IPQ807x	Yes
DIGEST	SHA1_HMAC	IPQ806x, IPQ807x	No
DIGEST	SHA256_HMAC	IPQ806x, IPQ807x	No
DIGEST	MD5_HMAC	IPQ807x	No
DIGEST	AES128_GMAC	IPQ807x	No
DIGEST	AES256_GMAC	IPQ807x	No
DIGEST	AES128_CMAC	IPQ807x	No

DIGEST	AES256_CMAC	IPQ807x	No
ABLK	AES256_CTR	IPQ806x, IPQ807x	Yes
ABLK	3DES_CBC	IPQ806x, IPQ807x	Yes
DIGEST	SHA1_HMAC	IPQ806x, IPQ807x	No
DIGEST	SHA256_HMAC	IPQ806x, IPQ807x	No
DIGEST	MD5_HMAC	IPQ807x	No
DIGEST	AES128_GMAC	IPQ807x	No
DIGEST	AES256_GMAC	IPQ807x	No
DIGEST	AES128_CMAC	IPQ807x	No
DIGEST	AES256_CMAC	IPQ807x	No

### 6.8.5 Restrictions

The current release of the crypto driver has the following limitations:

- Support for scatter gather data buffers is not available in the release and hence all data buffers are assumed to be in contiguous memory
- Hash generated is assumed to be inside the data buffer and data buffers must have at least 128 bytes of free space available from the location where hash has to be generated by HW
- Until QCA\_Networking\_2016.SP4.0 release, a maximum of 16 sessions was supported. Starting with QCA\_Networking\_2017.SP5.0 release, the maximum number of supported crypto sessions is increased to 64. Out of the 64 crypto sessions, first 16 will be allocated in OCM and the rest in DDR. Thus, the throughput and performance will reduce once the crypto sessions go beyond the 16 in OCM.

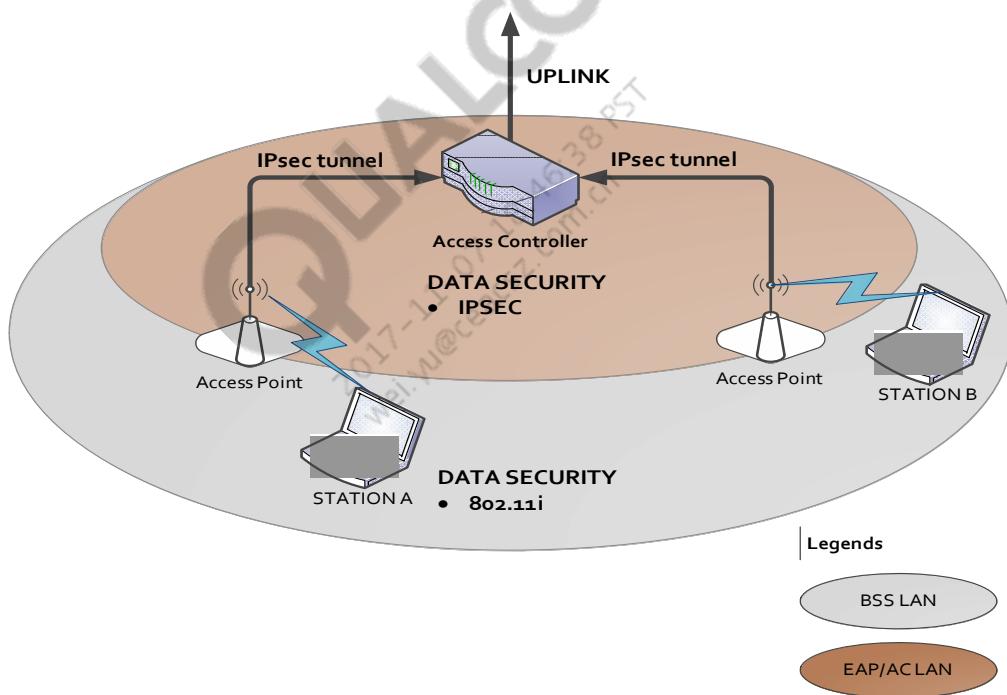
## 6.9 IPsec Tunnels

IPsec secures a communication channel between a sender and receiver of the data. The primary objective for a secure IPsec link is to:

- Correctly decode the data that was sent
- Validate that it originated from the trusted sender
- The sender must negotiate keys and algorithms between it and the receiver such that the message encoded by sender can only be decoded by the receiver. Also, the receiver must get enough information in the message so that it can validate its origin. IPsec acts a secure tunnel inside which Layer-2 frames or Layer-3 frames can be securely transported.

IPsec can be deployed in various configurations; this section shows the most popular.

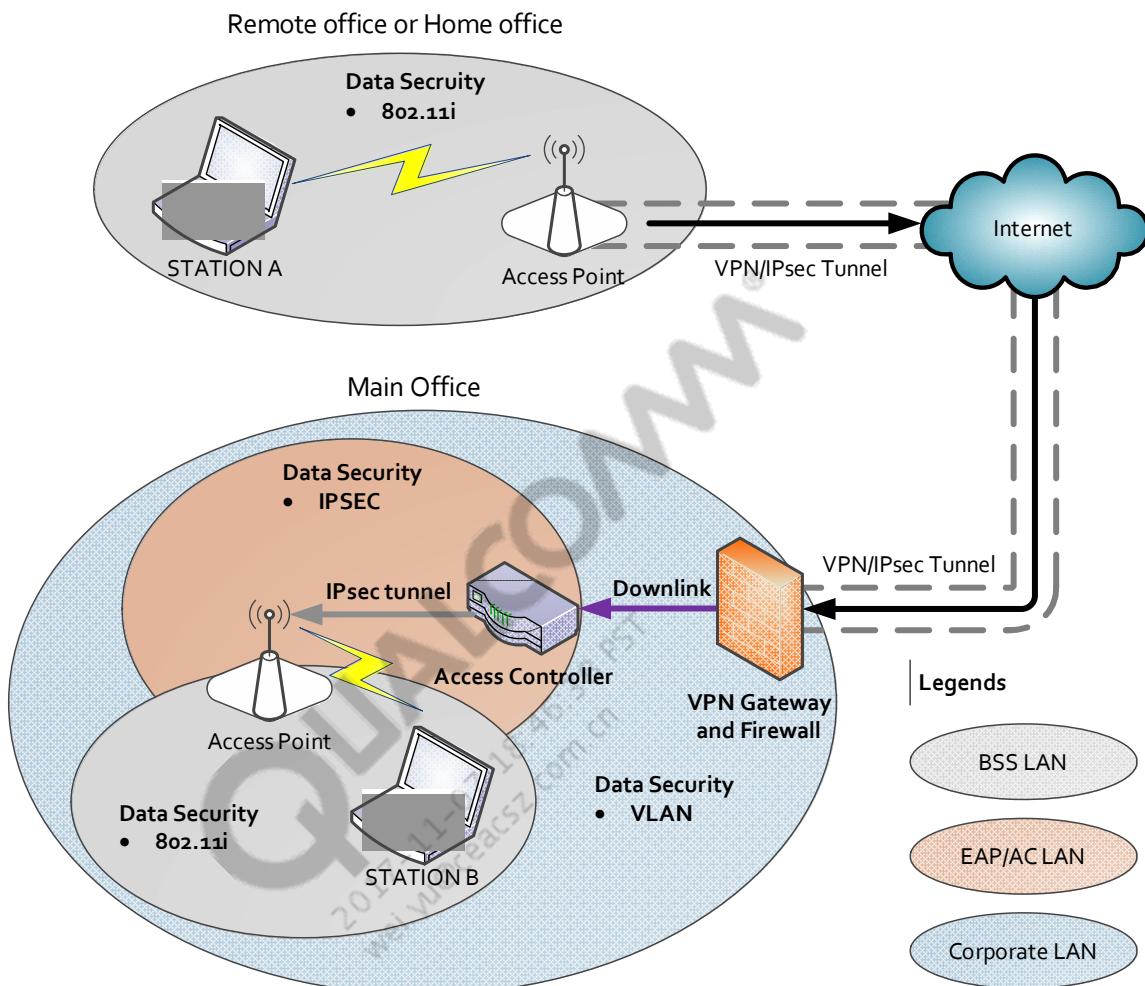
### 6.9.1 Campus deployment



**Figure 6-5 Campus Deployment using IPsec**

In the campus deployment a Wireless AP will communicate to the Access Controller using the secure IPsec tunnel. All traffic received from its associated stations at the Access Point are encapsulated and transformed into a secure payload which is sent to the Access Controller for any forwarding decision.

## 6.9.2 Remote branch office



**Figure 6-6 Remote branch office deployment**

In this deployment the remote AP will initiate a VPN/IPsec tunnel to transport the data from its BSS towards the Main office and use IPsec to secure the payload exchange between the two sites. This way a small group of user(s) can remotely work with the main office without the need for initiating VPN connections to main office's VPN gateway.

### 6.9.3 IPsec with crypto offload

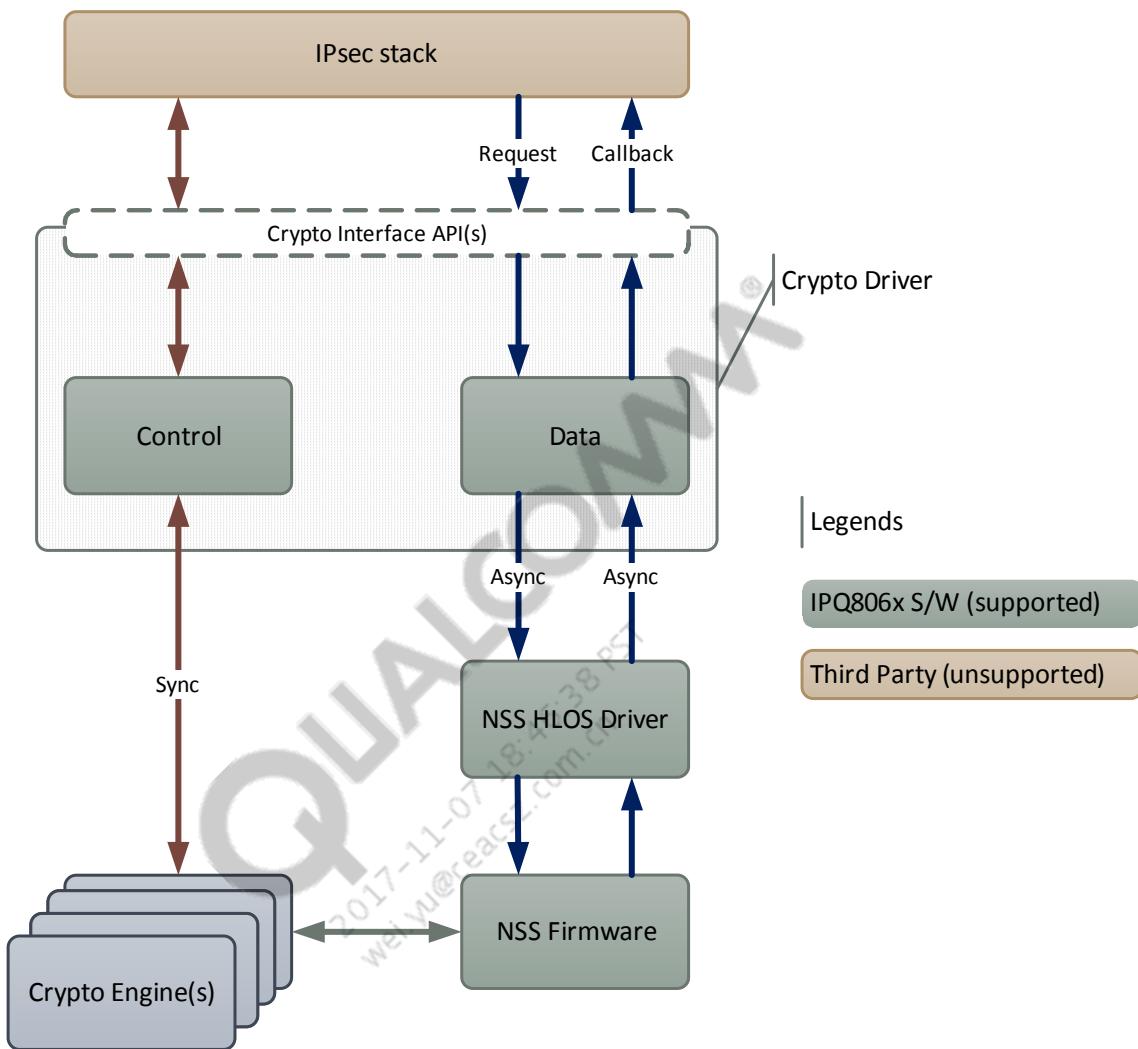


Figure 6-7 Crypto offload in context of IPsec

The preceding figure explains the organization of the crypto driver and its interaction in the context of an IPsec stack. All synchronous requests like session allocation, buffer allocation are handled by the control component and the asynchronous requests such as transform payload are handled by the data component. The control component directly interacts with HW when required but data component always interacts with the NSS firmware through the NSS-HLOS driver, which helps in segregating bulk of processing to the firmware and provides a lightweight driver in the host.

## 6.10 IPsec Acceleration

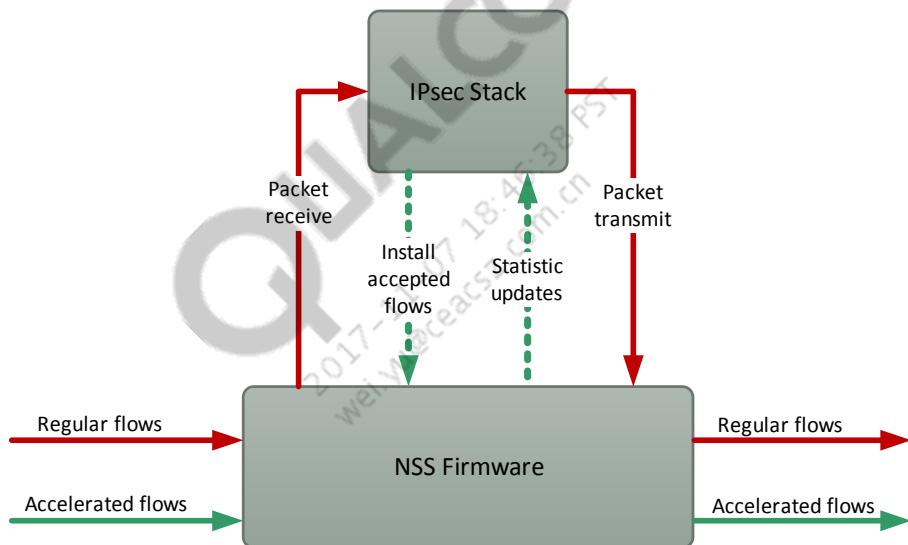
One of the key requirements for IPsec stack is to maintain the incoming packet rate between the unmanaged ports and tunnel ports. This requires the IPsec stack to have lower latency and high efficiency in its data path while modifying every single packet. Most IPsec stacks are bulkier and have to perform lot of tasks in the data path before allowing the traffic to flow through the tunnel.

But once a flow has been accepted it doesn't require the extensive set of checks for doing encapsulation or decapsulation.

### 6.10.1 Acceleration model

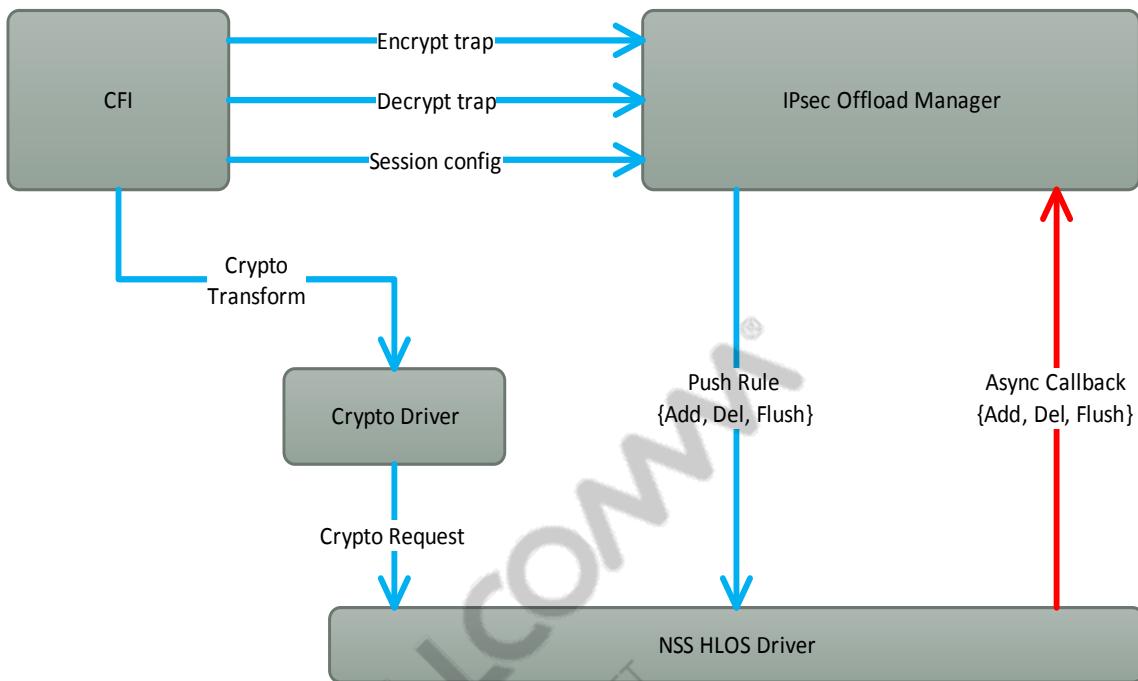
Using this fact as the basis, acceleration can be done for those flows which have been validated and accepted by the IPsec stack. This section summarizes the advantages of offloading:

- CPU-intensive operations such as header encapsulation, hash comparison, crypto operations, etc., are executed by the firmware instead of Host IPsec stack.
- Only flows that have been accepted by the IPsec stack and have passed all policy checks (pre-encryption and post-decryption) will get installed for acceleration, thus retaining control over the flows from host
- Zero CPU cost on Host for dropping packets after hash failure
- Firmware supporting acceleration also enables crypto offload for non-accelerated flows



**Figure 6-8 IPsec acceleration model for IPQ806x**

IPQ806x Enterprise firmware is enabled with acceleration so it can be tested with an IPsec stack. With the help of acceleration management SW provided with the release it can be enabled to push flows which have been validated and accepted for tunneling, after which all consecutive flows will move directly through the NSS and will not see the host under normal circumstances. This mechanism can be tested in the current release so that a measure of performance improvements over the regular IPsec path can be seen.



**Figure 6-9 IPsec offload configuration of NSS firmware**

IPsec offload manager traps frames coming for crypto transforms, in this case from CFI and processes its headers for creating rules needed in IPsec accelerated path. It relies completely on the Host IPsec stack for identifying the changes needed on a frame using a particular SA. It then prepares a rule and the associated data that the accelerated path should use for IPsec, and pushes it down to the NSS firmware.

For each of these rules the insertions and deletion from the IPsec Accelerated path returns a sync message as an acknowledgment for the operation. The IPsec offload manager maintains a local state to validate these and update its local state with appropriate entries.

It also traps configuration changes like session deletion for clearing the associated rules with a particular SA (Security Association). This flushing is asynchronous and synchronous from the perspective of the Offload Manager, depending upon whether it is flushing a single SA or all SA(s) from the IPsec accelerated path.

## 6.10.2 APIs

IPsec offload manager is responsible for accelerating the flows accepted by the IPsec stack running on host. It uses the following APIs to push rules into the firmware for IPsec acceleration. Also, it receives various IPsec events from firmware in lieu of the pushed rules or as an independent sync message for statistics updates. This section covers some of the Key APIs that should be used for IPsec acceleration; for the complete list of APIs and its description can be found in API documentation.

### 6.10.2.1 Creating/Adding IPsec tunnel

Name	<code>nss_ipsecmgr_tunnel_add</code>
Input	Ipsec Manager Callback information
Output	Linux Net Device or NULL

This API creates an IPsec pseudo tunnel device. It also registers notification callback handler for IPsec encap and decap interface for this tunnel device with NSS FW.

#### 6.10.2.2 Destroying/Deleting IPsec tunnel

Name	<code>nss_ipsecmgr_tunnel_del</code>
Input	Linux Net Device
Output	Success or Failure

This API deregisters all the notification handlers associated with the tunnel device.

#### 6.10.2.3 Adding Encap flow rule

Name	<code>nss_ipsecmgr_encap_add</code>
Input	IPsec Tunnel device, Flow information, Security Association for the flow, additional SA data
Output	Success or Failure

This API adds Encap flow rule to the IPsec offload database. Packets matching the flow information will be offloaded for IPsec encapsulation.

The Flow information [security policy] contains the selector information of packets for which the SA has to be applied. Flow information will usually be three tuples Source IP, Destination IP and Protocol.

Security Association [SA] contains the tunnel information such as Outer Source IP, Outer Destination IP and SPI index.

SA data has the additional information to perform the cipher and authentication operations. Usually Crypto index, ICV length, anti-replay window size, etc.

#### 6.10.2.4 Delete Encap flow rule

Name	<code>nss_ipsecmgr_encap_del</code>
Input	IPsec tunnel device, Flow information, Security Association for the flow
Output	Success or Failure

This API deletes Encap flow rule from the IPsec offload database. Once the rule is deleted packets will no longer be processed in the fast path and will be handled by the host stack.

The flow rule matching selector information and SA will be deleted from the offload database. If the match is not found, it will return failure.

### 6.10.2.5 Add Decap flow rule

Name	<code>nss_ipsecmgr_decap_add</code>
Input	IPsec tunnel device, Security Association for the flow, additional SA data
Output	Success or Failure

This API adds Decap flow rule to the IPsec offload database. Packets matching the SA information will be offloaded for IPsec decapsulation.

Security Association [SA] contains the tunnel information such as Outer Source IP, Outer Destination IP, and SPI index.

SA data has the additional information to perform the cipher and authentication operations. Usually Crypto index, ICV length, anti-replay window size, etc.

### 6.10.2.6 Flush SA

Name	<code>nss_ipsecmgr_sa_flush</code>
Input	IPsec tunnel device, Security Association for the flow
Output	Success or Failure

This API flushes the SA and all associated flows.

Security association [SA] contains the tunnel information such as Outer Source IP, Outer Destination IP, and SPI index.

### 6.10.3 Restrictions

This acceleration mechanism has the following limitations:

- These APIs are available post IPQ8064.ILQ.3.1.r2/000000032.1
- Support for handling sequence number wraparound condition is unavailable in the current release; future software releases will support this
- Support for fragmentation and reassembly in accelerated path is unavailable hence MTU at the traffic source and sink must be reduced to test this configuration.

## 6.11 IPsec Bring Up

IPsec can be tested using various test topologies, but the two mechanisms described in this section are simplest and most effective way of verifying its performance and characteristics:

- Connecting a IPsec enabled router (based on IPQ806x) to an IPsec capable traffic generator (Ixia Xcellon Ultra-XTS)
- Connecting ‘2’ IPsec enabled router (based on IPQ806x) in a back to back mode and attaching traffic generators to the respective back ends.

The preferred mechanism is (1) because it provides a large set of statistics at an IPsec level and independently validates the incoming packets while decrypting it as a tunnel termination endpoint.

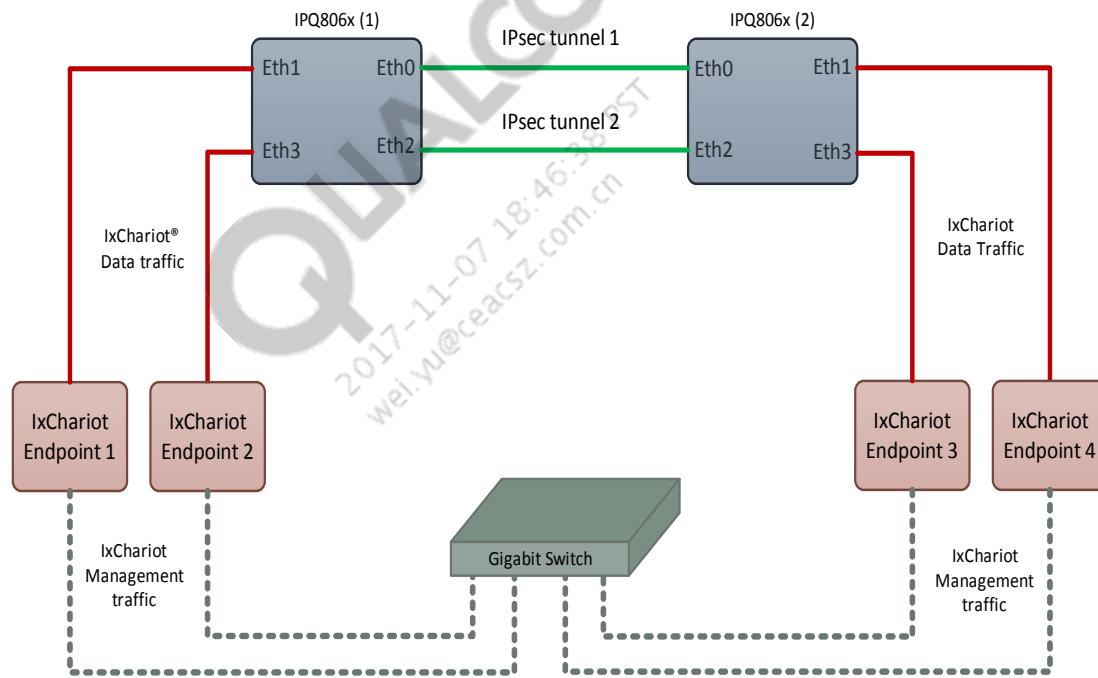
But, (2) is an inexpensive and non-device specific way of performing the same level of testing with certain constraints such as increase in latency of the packet path in case of TCP and unavailability of IPsec specific statistics at the traffic termination points.

The following sections can be used to create an IPsec topology for demonstration. The choice of topology is based upon the reference boards that are being used. An IPsec setup can be demonstrated for

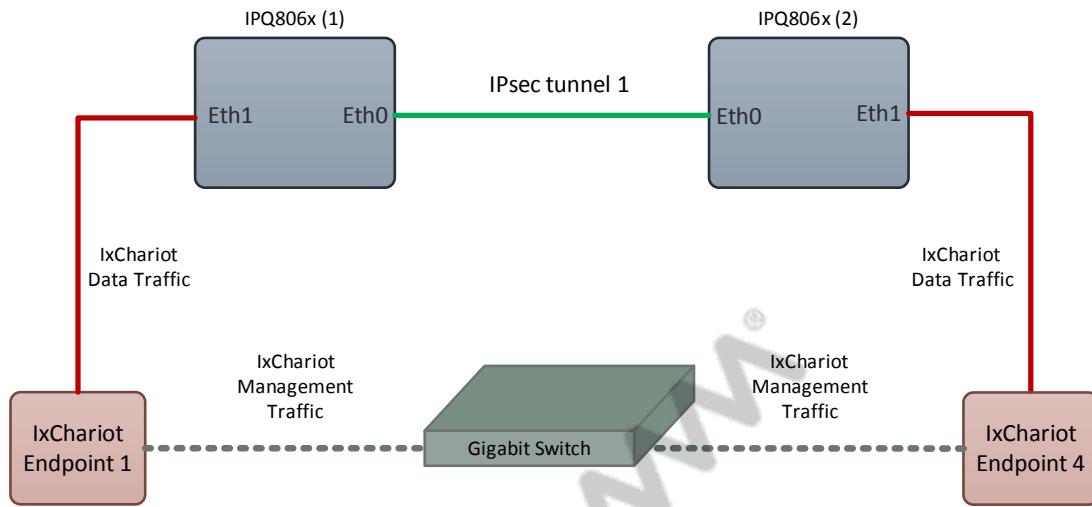
- Two tunnels using reference boards having 4 GMAC ports (e.g., DB149)
- Or one tunnel using reference boards having 2 GMAC ports (e.g., AP148)

### 6.11.1 IPsec setup

This section describes a back-to-back setup using two IPQ806x based boards. The following figures show two IPQ806x based boards connected in a back-to-back manner for demonstrating IPsec test topology in a 2 tunnel setup and in a 1-tunnel setup.



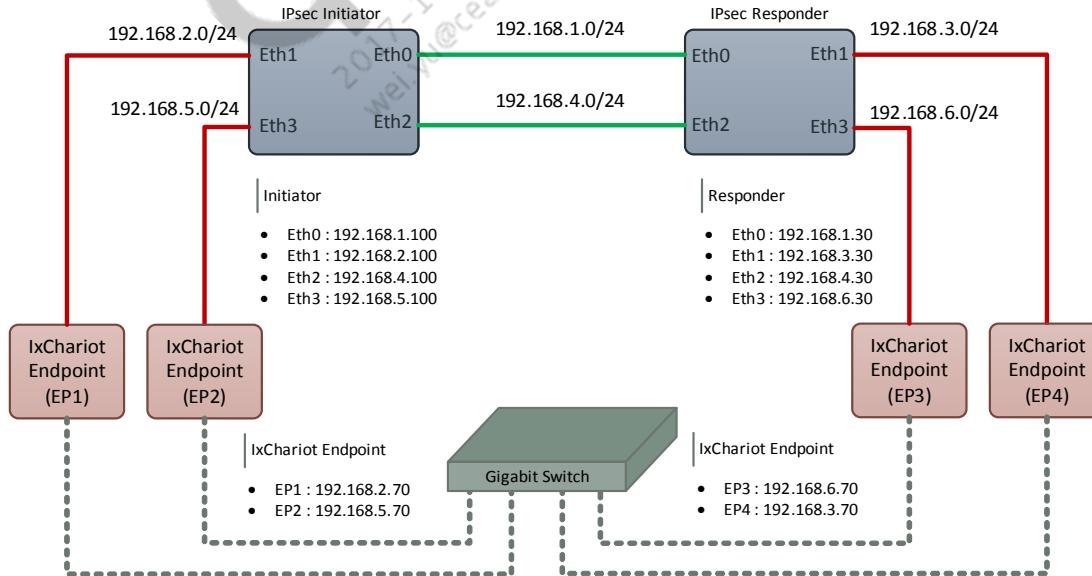
**Figure 6-10 Connection topology for a “2” tunnel IPsec setup**



**Figure 6-11 Connection topology for a “1” tunnel IPsec setup**

The preceding two figures describe the network topology for the back-to-back connection; it is required to make sure that the IPQ806x (1 & 2) behaves as a router and routes traffic between the back ends connected behind them. It is assumed that IPQ806x (1) will be an IPsec initiator and IPQ806x (2) will be an IPsec responder. This document refers to them as initiator and responder.

The role of the initiator is to start the IPsec connection request and the role of the responder is to add new tunnel requests arriving from initiators. It is necessary to have the IPsec daemon started on the responder before starting the IPsec daemon on the initiator.



**Figure 6-12 Subnets and IP(s) used in the “2” tunnel IPsec topology**

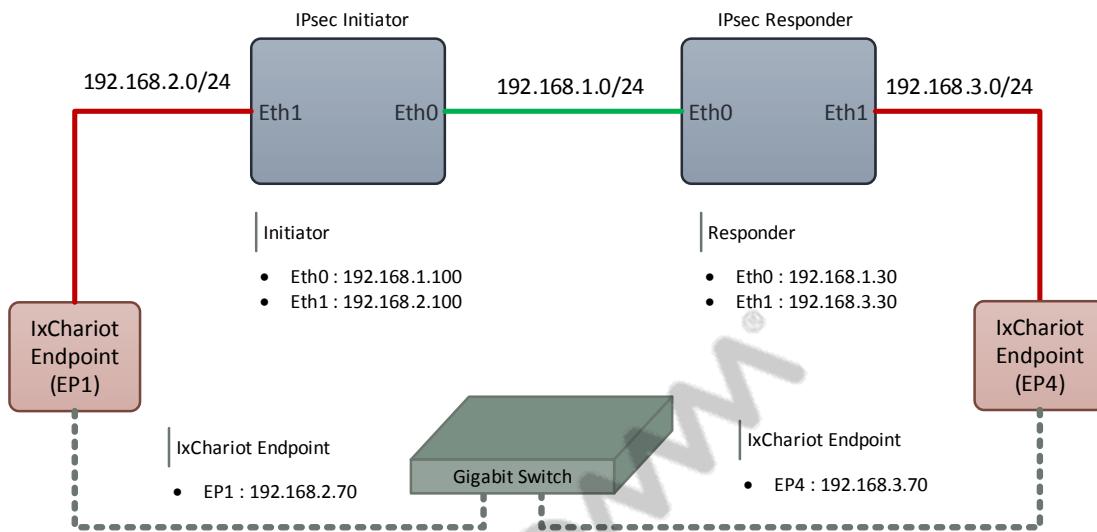


Figure 6-13 Subnets and IP(s) used in the “1” tunnel IPsec topology

### 6.11.2 MTU

The IPsec tunnel will add extra headers to the packet during tunneling operation. It is mandatory to reduce the MTU of the traffic generator if fragmentation at the router needs to be avoided which affects performance. The packet coming out of the traffic generator should not exceed more than 1408 bytes as the IPsec tunnel header will have an additional 100 bytes to add to the packet. The tunneled traffic must not exceed the Ethernet MTU or else fragmentation may occur at either routers.

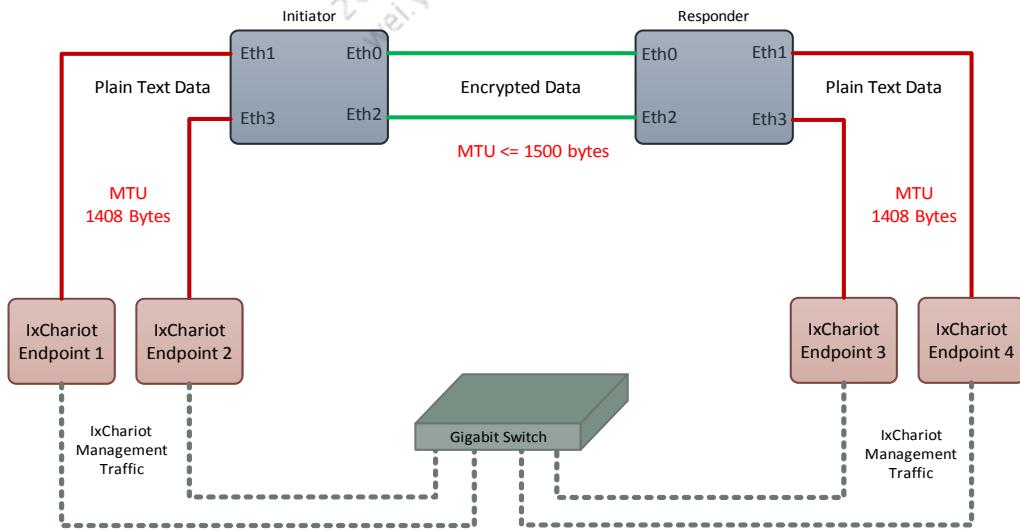


Figure 6-14 Traffic pattern and the corresponding MTU(s)

### 6.11.3 IPsec Bring up (LAN to WAN)

The following is the prerequisite for bringing up IPsec fast path

Flash enterprise single image on the IPQ806x boards using the flashing instructions for IPQ806x. The IPsec setup requires different configuration on Initiator and Responder. This section details the common commands required to be executed on both.

The firewall needs to be always disabled before configuring IPsec tunnel.

```
/etc/init.d/firewall disable  
/etc/init.d/firewall stop
```

### 6.11.3.1 Netkey

While configuring IPsec tunnel in netkey stack, only crypto transformation is offloaded to NSS and not the IPsec flow. Hence, we need to disable ECM before the IPsec configuration.

```
/etc/init.d/qca-nss-ecm stop
```

### 6.11.3.2 Initiator

#### Network Configuration

```
uci set network.lan=interface  
uci set network.lan.ifname='eth1'  
uci set network.lan.type=bridge  
uci set network.lan.proto=static  
uci set network.lan.ipaddr=192.168.2.100  
uci set network.lan.netmask=255.255.255.0  
uci set network.wan=interface  
uci set network.wan.ifname=eth0  
uci set network.wan.proto=static  
uci set network.wan.ipaddr=192.168.1.100  
uci set network.wan.netmask=255.255.255.0  
uci commit  
/etc/init.d/network restart
```

#### IPsec Configuration

```
uci set ipsec.setup.nat_traversal=no  
uci set ipsec.setup.protostack=klips [required only for KLIPS]  
uci set ipsec.setup.interfaces="ipsec0=eth0" [required only for KLIPS]  
uci set ipsec.setup.protostack=netkey [required only for NETKEY]  
  
uci set ipsec.setup.oe=off  
uci set ipsec.wan_wan_1=ipsec_conn  
uci set ipsec.wan_wan_1.authby=secret  
uci set ipsec.wan_wan_1.type=tunnel  
uci set ipsec.wan_wan_1.pfs=yes
```

```

uci set ipsec.wan_wan_1.rekey=yes
uci set ipsec.wan_wan_1.ikelifetime=24h
uci set ipsec.wan_wan_1.salifetime=24h
uci set ipsec.wan_wan_1.left=192.168.1.100
uci set ipsec.wan_wan_1.leftsubnet=192.168.2.0/24
uci set ipsec.wan_wan_1.leftnexthop=192.168.1.30
uci set ipsec.wan_wan_1.right=192.168.1.30
uci set ipsec.wan_wan_1.rightsubnet=192.168.3.0/24
uci set ipsec.wan_wan_1.rightnexthop=192.168.1.100
uci set ipsec.wan_wan_1.ikev2=insist
uci set ipsec.wan_wan_1.ike=aes128-sha1
uci set ipsec.wan_wan_1.esp=aes128-sha1
uci set ipsec.wan_wan_1.auto=start

uci add ipsec ipsec_secret_ss
uci set ipsec.@ipsec_secret_ss[-1].secret="ipsec"
uci set ipsec.@ipsec_secret_ss[-1].indices="%any %any"
uci commit
/etc/init.d/ipsec restart
route add -net 192.168.3.0/24 gw 192.168.1.30 [Required only for NETKEY]

```

### 6.11.3.3 Responder

#### Network Configuration

```

uci set network.lan=interface
uci set network.lan.ifname='eth1'
uci set network.lan.type=bridge
uci set network.lan.proto=static
uci set network.lan.ipaddr=192.168.3.30
uci set network.lan.netmask=255.255.255.0
uci set network.wan=interface
uci set network.wan.ifname=eth0
uci set network.wan.proto=static
uci set network.wan.ipaddr=192.168.1.30
uci set network.wan.netmask=255.255.255.0
uci commit
/etc/init.d/network restart

```

#### IPsec Configuration

```

uci set ipsec.setup.nat_traversal=no
uci set ipsec.setup.protostack=klips [required only for KLIPS]
uci set ipsec.setup.interfaces="ipsec0=eth0" [required only for KLIPS]

```

```

uci set ipsec.setup.protostack=netkey [required only for NETKEY]
uci set ipsec.setup.oe=off
uci set ipsec.wan_wan_1=ipsec_conn
uci set ipsec.wan_wan_1.authby=secret
uci set ipsec.wan_wan_1.type=tunnel
uci set ipsec.wan_wan_1.pfs=yes
uci set ipsec.wan_wan_1.rekey=yes
uci set ipsec.wan_wan_1.ikelifetime=24h
uci set ipsec.wan_wan_1.salifetime=24h
uci set ipsec.wan_wan_1.left=192.168.1.100
uci set ipsec.wan_wan_1.leftsubnet=192.168.2.0/24
uci set ipsec.wan_wan_1.leftnexthop=192.168.1.30
uci set ipsec.wan_wan_1.right=192.168.1.30
uci set ipsec.wan_wan_1.rightsubnet=192.168.3.0/24
uci set ipsec.wan_wan_1.rightnexthop=192.168.1.100
uci set ipsec.wan_wan_1.ikev2=insist
uci set ipsec.wan_wan_1.ike=aes128-sha1
uci set ipsec.wan_wan_1.esp=aes128-sha1
uci set ipsec.wan_wan_1.auto=add
uci add ipsec ipsec_secret_ss
uci set ipsec.@ipsec_secret_ss[-1].secret="ipsec"
uci set ipsec.@ipsec_secret_ss[-1].indices="%any %any"
uci commit
/etc/init.d/ipsec restart
route add -net 192.168.2.0/24 gw 192.168.1.100 [Required only for NETKEY]

```

#### 6.11.3.4 Common commands

```
/etc/init.d/ipsec restart
ipsec eroute
```

Initiate pings from the IxChariot endpoints EP1 → EP4. This step is mandatory for resolving the ARPs, without which traffic cannot be initiated.

Once the pings start to work, IxChariot traffic can be initiated using two pairs of high performance throughput scripts from the IxChariot console.

#### 6.11.4 IPsec bring up (WLAN to WAN)

The following figure shows the connection topology for a WLAN-to-WAN IPsec bring up. The following table shows the network configuration.

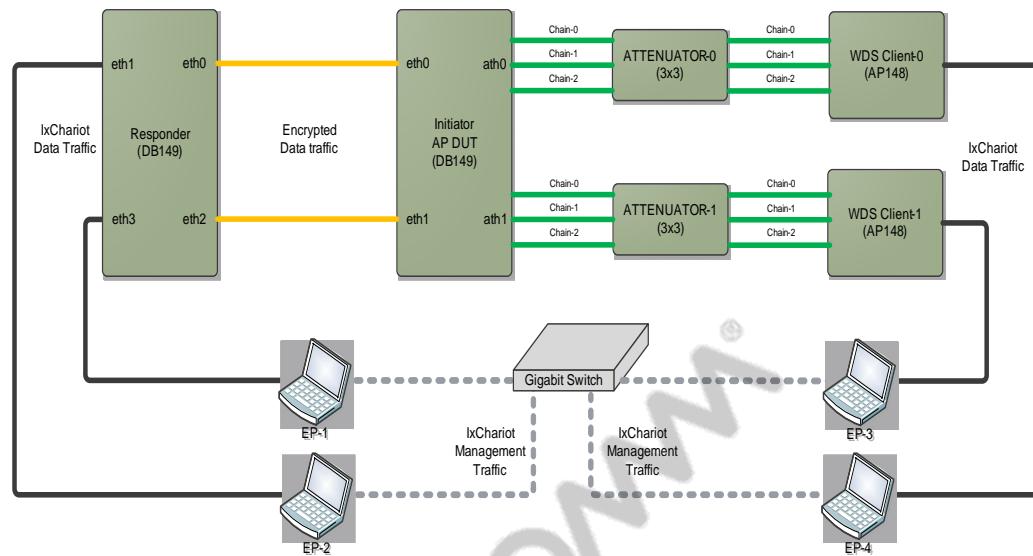


Figure 6-15 Connection topology for a “2” tunnel IPsec + WLAN setup

Table 6-1 Network configuration

Interface/DUT	Responder	Initiator	WDS Client – 0	WDS Client – 1
ETH0	192.168.1.30	192.168.1.100	--	--
ETH1	192.168.3.30	--	--	--
ETH2	192.168.4.30	192.168.4.100	--	--
ETH3	192.168.6.30	--	--	--
ATH0	--	192.168.2.100	--	--
ATH1	--	192.168.5.100	--	--
BR-LAN	--	--	192.168.8.1	192.168.9.1

### Prerequisite for bringing up IPsec fast path

Flash Enterprise single image on the IPQ806x boards using the flashing instructions for IPQ806x. Perform the following steps to run the IPsec fast path:

1. IPQ806x(s) must be connected as per the topology mentioned in preceding figure , it is recommended to use the ports shown.
2. Recommended IP address, on the corresponding DUT(s) should be used from the preceding table.
3. The IP addresses on WDS Clients must be changed to the mentioned ones in preceding table. Otherwise, the basic routing will not work due to the default IP(s) of these clients.
4. Enable IPsec by following the procedure shown in the *IPsec Bring up (LAN to WAN)* section, except step 3.
5. Enable Wi-Fi.

6. Start IxChariot traffic using the recommended scripts for running WLAN throughput.

### 6.11.5 Verify NSS IPsec features

This section details the steps to be followed to verify the features supported by NSS IPsec.

#### 6.11.5.1 Max crypto sessions

Presently, the maximum number of crypto sessions supported is 64. This can be tested irrespective of configuring IPsec in KLIPS or netkey stack. The following steps should be followed to test the support of maximum possible crypto sessions.

1. Set the session delete timeout to a high value; say 20 mins.  
`echo 1200 > /sys/module/qca_nss_crypto/parameters/free_timeout`
2. Configure IPsec tunnel with a low re-key timeout; say 3 mins.  
`uci set ipsec.wan_wan_1.ikelifetime=12m  
uci set ipsec.wan_wan_1.salifetime=3m`
3. Run traffic
4. Stats about the number of crypto sessions allocated, freed or failed can be obtained through commands mentioned later in this section.

### 6.11.6 AES-192 software fallback

IPQ8064 does not support crypto offload for AES-192 cipher algorithm and hence a software fallback has been devised to support this. This can be tested following the given steps

1. Stop ECM.  
`/etc/init.d/qca-nss-ecm stop`
2. This can be only tested by configuring IPsec in netkey stack.
3. For testing AEAD process software fallback, configure on both initiator and responder:  
`uci set ipsec.wan_wan_1.esp=aes192-sha1`
4. For testing ABLK process software fallback, configure on both initiator and responder:  
`uci set ipsec.wan_wan_1.esp=aes192-md5`
5. Run traffic.

## 6.11.7 AES-CTR

Openswan and KLIPS stack do not support AES-CTR cipher algorithm for cryptographic operations. Therefore, use ip xfrm utility to set the key for IPsec tunnel in netkey stack. Here are the common commands that need to be run on both initiator and responder:

1. Stop firewall  
`/etc/init.d/firewall stop`
1. Stop ECM.  
`/etc/init.d/qca-nss-ecm stop`
2. Flush the IPsec XFRM states and policies.  
`ip xfrm policy flush`  
`ip xfrm state flush`

The rest of the commands need to be run on initiator or responder accordingly as outlined in the following sections.

## 6.11.8 Initiator

### Network Configuration

```
uci set network.lan=interface
uci set network.lan.ifname='eth1'
uci set network.lan.type=bridge
uci set network.lan.proto=static
uci set network.lan.ipaddr=192.168.2.100
uci set network.lan.netmask=255.255.255.0
uci set network.wan=interface
uci set network.wan.ifname=eth0
uci set network.wan.proto=static
uci set network.wan.ipaddr=192.168.1.100
uci set network.wan.netmask=255.255.255.0
uci commit
/etc/init.d/network restart
```

### IPsec Configuration

AES-CTR can be configured either in AEAD mode; with SHA1 authentication or in ABLK mode with MD5 authentication algorithm.

#### AEAD AES-CTR

```
ip xfrm state add src 192.168.1.100 dst 192.168.1.30 \
proto esp spi 34502 \
mode tunnel \
enc "rfc3686(ctr(aes))" 0x7D8ADC5F52E6B2291F27A4C0384624A7A1B2C3D4 \
\
```

```

auth "hmac(shal)" 0x8D967D88F6CAA9D714800AB3D48051D63F73A312 \
      ip xfrm state add src 192.168.1.30 dst 192.168.1.100 \
          proto esp spi 34502 \
          mode tunnel \
          enc "rfc3686(ctr(aes))" 0x7D8ADC5F52E6B2291F27A4C0384624A7A1B2C3D4 \
      \
      auth "hmac(shal)" 0x8D967D88F6CAA9D714800AB3D48051D63F73A312 \
      \
          ip xfrm policy add src 192.168.3.0/24 dst 192.168.2.0/24 dir
in \
          tmpl src 192.168.1.30 dst 192.168.1.100 proto esp mode
tunnel

      ip xfrm policy add src 192.168.2.0/24 dst 192.168.3.0/24 dir out \
          tmpl src 192.168.1.100 dst 192.168.1.30 proto esp mode tunnel

      ip xfrm policy add src 192.168.3.0/24 dst 192.168.2.0/24 dir fwd \
          tmpl src 192.168.1.30 dst 192.168.1.100 proto esp mode
tunnel

```

### **ABLK AES-CTR**

```

ip xfrm state add src 192.168.1.100 dst 192.168.1.30 \
      proto esp spi 34502 \
      mode tunnel \
      enc "rfc3686(ctr(aes))" 0x7D8ADC5F52E6B2291F27A4C0384624A7A1B2C3D4 \
      \
      auth "md5" 0x8D967D88F6CAA9D714800AB3D48051D63F73A312 \
      \
          ip xfrm state add src 192.168.1.30 dst 192.168.1.100 \
              proto esp spi 34502 \
              mode tunnel \
              enc "rfc3686(ctr(aes))" 0x7D8ADC5F52E6B2291F27A4C0384624A7A1B2C3D4 \
      \
          auth "md5" 0x8D967D88F6CAA9D714800AB3D48051D63F73A312 \
      \
              ip xfrm policy add src 192.168.3.0/24 dst 192.168.2.0/24 dir
in \
              tmpl src 192.168.1.30 dst 192.168.1.100 proto esp mode
tunnel

          ip xfrm policy add src 192.168.2.0/24 dst 192.168.3.0/24 dir out \
              tmpl src 192.168.1.100 dst 192.168.1.30 proto esp mode tunnel

```

```
ip xfrm policy add src 192.168.3.0/24 dst 192.168.2.0/24 dir fwd \
    tmpl src 192.168.1.30 dst 192.168.1.100 proto esp mode
tunnel
```

## 6.11.9 Responder

### Network Configuration

```
uci set network.lan=interface
uci set network.lan.ifname='eth1'
uci set network.lan.type=bridge
uci set network.lan.proto=static
uci set network.lan.ipaddr=192.168.3.30
uci set network.lan.netmask=255.255.255.0
uci set network.wan=interface
uci set network.wan.ifname=eth0
uci set network.wan.proto=static
uci set network.wan.ipaddr=192.168.1.30
uci set network.wan.netmask=255.255.255.0
uci commit
```

/etc/init.d/network restart

### IPsec Configuration

AES-CTR can be configured either in AEAD mode with SHA1 authentication or in ABLK mode with MD5 authentication algorithm.

#### AEAD AES-CTR

```
ip xfrm state add src 192.168.1.30 dst 192.168.1.100 \
    proto esp spi 34502 \
    mode tunnel \
    enc "rfc3686(ctr(aes))" 0x7D8ADC5F52E6B2291F27A4C0384624A7A1B2C3D4 \
    \
    auth "hmac(shal)" 0x8D967D88F6CAA9D714800AB3D48051D63F73A312 \

ip xfrm state add src 192.168.1.100 dst 192.168.1.30 \
    proto esp spi 34502 \
    mode tunnel \
    enc "rfc3686(ctr(aes))" 0x7D8ADC5F52E6B2291F27A4C0384624A7A1B2C3D4 \
    \
    auth "hmac(shal)" 0x8D967D88F6CAA9D714800AB3D48051D63F73A312 \

ip xfrm policy add src 192.168.2.0/24 dst 192.168.3.0/24 dir in \
```

```

        tmpl src 192.168.1.100 dst 192.168.1.30 proto esp mode
tunnel

ip xfrm policy add src 192.168.3.0/24 dst 192.168.2.0/24 dir out \
tmpl src 192.168.1.30 dst 192.168.1.100 proto esp mode tunnel

ip xfrm policy add src 192.168.2.0/24 dst 192.168.3.0/24 dir fwd \
tmpl src 192.168.1.100 dst 192.168.1.30 proto esp mode
tunnel

```

### **ABLK AES-CTR**

```

ip xfrm state add src 192.168.1.30 dst 192.168.1.100 \
proto esp spi 34502 \
mode tunnel \
enc "rfc3686(ctr(aes))" 0x7D8ADC5F52E6B2291F27A4C0384624A7A1B2C3D4
\
auth "md5" 0x8D967D88F6CAA9D714800AB3D48051D63F73A312 \

ip xfrm state add src 192.168.1.100 dst 192.168.1.30 \
proto esp spi 34502 \
mode tunnel \
enc "rfc3686(ctr(aes))" 0x7D8ADC5F52E6B2291F27A4C0384624A7A1B2C3D4
\
auth "md5" 0x8D967D88F6CAA9D714800AB3D48051D63F73A312 \

ip xfrm policy add src 192.168.2.0/24 dst 192.168.3.0/24 dir in \
tmpl src 192.168.1.100 dst 192.168.1.30 proto esp mode
tunnel

ip xfrm policy add src 192.168.3.0/24 dst 192.168.2.0/24 dir out \
tmpl src 192.168.1.30 dst 192.168.1.100 proto esp mode tunnel

ip xfrm policy add src 192.168.2.0/24 dst 192.168.3.0/24 dir fwd \
tmpl src 192.168.1.100 dst 192.168.1.30 proto esp mode
tunnel

```

## **6.11.10 Statistics**

### **6.11.10.1 Tunnel Statistics**

Over-all IPsec tunnel statistics can be retrieved from the following command.

```
root@OpenWrt:/# ifconfig ipsec tunx
```

where x, can be 0, 1, 2... 7[max]

This command provides a count of transmitted, received and dropped packets of a particular IPsec tunnel.

### 6.11.10.2 Crypto Statistics

The crypto statistics can be accessed using the debugfs interface. Statistics can be acquired on a per-engine and per-active crypto session basis. Crypto statistics can be accessed via flowing debugfs entries.

In these tables:

- (engineX): x is the engine number (0, 1, 2, 3 for IPQ806X)
- (sessionY) y is the session number [0-15]

The entry is present only for active sessions.

**Table 6-2 Cumulative crypto statistics**

Debugfs Entry	Description
/sys/kernel/debug/qca-nss-crypto/stats/total/queued	Number of packets queued for Cipher operation.
/sys/kernel/debug/qca-nss-crypto/stats/total/completed	Number of packets for which Cipher operation is complete.
/sys/kernel/debug/qca-nss-crypto/stats/total/dropped	Number of packets which were dropped during Cipher operation.

**Table 6-3 Engine level crypto statistics**

Debugfs Entry	Description
/sys/kernel/debug/qca-nss-crypto/stats/engineX/queued	Number of packets queued to engineX for Cipher operation.
/sys/kernel/debug/qca-nss-crypto/stats/engineX/completed	Number of packets queued to engineX for which Cipher operation is complete.
/sys/kernel/debug/qca-nss-crypto/stats/engineX/dropped	Number of packets queued to engineX which were dropped during Cipher operation.

**Table 6-4 Session Level crypto statistics**

Debugfs Entry	Description
/sys/kernel/debug/qca-nss-crypto/stats/session Y/queued	Number of packets queued to engineX for Cipher operation.

/sys/kernel/debug/qca-nss-crypto/stats/session Y/completed	Number of packets queued to engineX for which Cipher operation is complete.
/sys/kernel/debug/qca-nss-crypto/stats/session Y/dropped	Number of packets queued to engineX which were dropped during Cipher operation.

**Table 6-5** Crypto configuration information

Debugfs Entry	Description
/sys/kernel/debug/qca-nss-crypto/config/session Y/cipher/<algorithm>	Cipher algorithm
/sys/kernel/debug/qca-nss-crypto/config/session Y/auth/<algorithm>	Authentication algorithm

**Table 6-6** Crypto Control information

Debugfs Entry	Description
sys/kernel/debug/qca-nss-crypto/stats/control/session_alloc	Number of crypto sessions allocated
sys/kernel/debug/qca-nss-crypto/stats/control/session_free	Number of crypto sessions freed
sys/kernel/debug/qca-nss-crypto/stats/control/session_alloc_fail	Number of failures in allocating crypto sessions

## 6.12 Extended NSS signaling

The existing 802.11 standard does not support negotiation of NSS as a function of channel bandwidth. As a result, certain workarounds are required to be used to advertise separate NSS capabilities for 80 MHz and 160 MHz modes. Recently, IEEE has made certain amendments to the standard called *Extended NSS*, which refers to signaling support for 160 and 80+80 MHz channel bandwidths with NSS support that is different from the maximum NSS signaled to legacy 80 MHz devices. The operating mode notification (OMN) switching to and from 160/80+80 MHz is also part of extended NSS signaling.

**NOTE** The OMN might be present in the management frames. The operating mode indication (OMI) is similar to OMN, but it controls also UL MU transmissions and OMI information is transmitted in the MAC headers. The AP may not receive and store the information correctly from the MAC headers of the unassociated STA.

The following table describes the management frames and fields involved in extended NSS signaling for Beacon, Probe Request/Response, and Association Request/Response:

Subfield	Element	Field	Function
CCFS0	VHT Operation element	VHT Operation Info field	Signals the center frequency of the primary 80 MHz channel
CCFS1	VHT Operation element	VHT Operation Info field	Signals the center frequency of the secondary 80 MHz channel or of the 160 MHz channel when NSS support is at least VHT Max NSS
CCFS2	HT Operation element	HT Operation Info field	Signals the center frequency of the secondary 80 MHz channel or of the 160 MHz channel when NSS support is less than VHT Max NSS

The following table describes the Channel Center Frequency (CCF) fields and elements used in Beacon, Probe response, and Association Response:

Subfield	Element	Field	Function
Extended NSS BW Support	VHT Capabilities element	VHT Capabilities Info field	Signals the Extended NSS support at a device
Extended NSS BW Capable	VHT Capabilities element	Supported VHT-MCS and NSS Set field	Signals whether the device can interpret the Extended NSS signaling

The following table describes the Operating Mode Notification:

Subfield	Element	Field	Function
160/80+80 BW	OMN element	VHT Capabilities Info field	Signals whether the BSS switches to or from 160/80+80 MHz BSS bandwidth

The following behavioral changes are implemented for AP on host side:

- Configure AP as Extended NSS capable.
- AP indicates support for Extended NSS Signaling along with existing Proprietary IE in Beacon/Probe response.
- Set appropriate value for supported channel width field in VHT capability field.
- Only Extended NSS Signaling is sent in association response, if the peer is Extended NSS Signaling-capable. Otherwise, the AP continues with proprietary IE for QCA9984 STA.
- Host to send three peer NSS values for 80 MHz, 160 MHz and 80\_80 MHz to FW during association.

The following behavioral changes are implemented for STA on host side:

- Configure STA as Extended NSS capable.
- QCA9984 STA to indicate support for Extended NSS Signaling along with existing Proprietary IE in Probe request.
- Set an appropriate value for supported channel width filed in VHT capability field.
- STA to check for extended NSS support in the received beacon/probe and make decisions accordingly.
- STA to send association response with only Extended NSS Signaling if AP is Extended NSS Signaling-capable.

The following signaling flow is used to determine NSS support per bandwidth (BW):

- At association:
  - Determine the Rx and Tx supported MCS and NSS set per BW using
    - Supported Channel Width Set subfield (from VHT Capabilities element)
    - Extended NSS BW Support subfield (from VHT Capabilities element)
  - Execute subclause 10.7.12.1 (Rx) and 10.7.12.2 (Tx)
- After receipt of an OMN element or frame (can be repeated):
  - Determine the Rx and Tx supported MCS and NSS set per BW using the following parameters:
    - Supported Channel Width Set subfield (from VHT Capabilities element)
    - Extended NSS BW Support subfield (from VHT Capabilities element)
    - OMN Channel Width subfield (from OMN field)
    - OMN 160 MHz BW subfield (from OMN field)
  - Execute subclause 10.7.12.1 (Rx) and 10.7.12.2 (Tx)

The following is the BSS bandwidth signaling workflow:

- The BSS bandwidth is signaled through CCFS0, CCF1, and CCFS2
  - CCFS0
    - CCFS0 signals the center frequency of the 80 MHz channel containing the primary channel
  - CCFS1 and CCFS2
    - CCFS1 and CCFS2 signal the center frequency of a 160 MHz channel or a secondary 80 MHz channel
      - CCFS1 = CCFS2 = 0: 160 or 80+80 MHz BSS bandwidth currently not active
      - CCFS1 > 0: 160 or 80+80 currently active, with NSS support at least maximum VHT NSS
      - CCFS2 > 0: 160 or 80+80 currently active, with NSS support less than maximum VHT NSS

- CCFS1 and CCFS2 > 0: Shall not happen
- 160 or 80+80 is known by the delta between CCFS1 or CCFS2 and CCFS0
  - 40 MHz apart: 160 MHz BSS bandwidth currently active
  - >80 MHz apart: 80+80 MHz BSS bandwidth currently active

The following are the possible settings specified in standard for the amended fields:

- VHT Capabilities Information field

**Table 9-250—Setting of the Supported Channel Width Set subfield and Extended NSS BW Support subfield at a STA transmitting the VHT Capabilities Information field**

Transmitted VHT Capabilities Information field	NSS Support of STA transmitting the VHT Capabilities Information field as a function of the PPDU bandwidth ( $\times$ Max VHT NSS) (see requirements R1 and R2)						Location of 160 MHz channel center frequency if BSS bandwidth is 160 MHz	Location of 80+80 MHz center frequency if BSS bandwidth is 80+80 MHz
Supported Channel Width Set	Extended NSS BW Support	20 MHz	40 MHz	80 MHz	160 MHz	80+80 MHz		
0	0	1	1	1				
0	1	1	1	1	1/2		CCFS2	
0	2	1	1	1	1/2	1/2	CCFS2	CCFS2
0	3	1	1	1	3/4	3/4	CCFS2	CCFS2
1	0	1	1	1	1	1	CCFS1	
1	1	1	1	1	1	1/2	CCFS1	CCFS2
1	2	1	1	1	1	3/4	CCFS1	CCFS2
1	3	2	2	2	2	1	CCFS1	CCFS1
2	0	1	1	1	1	1	CCFS1	CCFS1
2	3	2	2	2	1	1	CCFS1	CCFS1

- OMN field

**Table 9-75—Setting of the Channel Width subfield and 160/80+80 BW subfield at a VHT STA transmitting the Operating Mode field**

Transmitted Operating Mode field		VHT Capabilities of STA transmitting the Operating Mode field						NSS Support of STA transmitting the Operating Mode field as a function of the PPDU bandwidth (<Max VHT NSS) (see requirements R1 and R2)		Location of 160 MHz center frequency if BSS bandwidth is 160 MHz		Location of secondary 80 MHz center frequency if BSS bandwidth is 80+80 MHz	
Channel width	160/80+80 BW	Supported Channel Width Set	Extended NSS BW Support	20 MHz	40 MHz	80 MHz	160 MHz	80 +80 MHz					
0	0	0-2	0-3	1									
1	0	0-2	0-3	1	1								
2	0	0-2	0-3	1	1	1							
2	1	0	1	1	1	1	1	1/2		CCFS2			
2	1	0	2	1	1	1	1	1/2	1/2	CCFS2	CCFS2		
2	1	0	3	1	1	1	3/4	3/4	3/4	CCFS2	CCFS2		
2	1	1	0	1	1	1	1			CCFS1			
2	1	1	1	1	1	1	1	1/2	1/2	CCFS1	CCFS2		
2	1	1	2	1	1	1	1	3/4	3/4	CCFS1	CCFS2		
2	1	1	3	2	2	2	2	1	1	CCFS1	CCFS1		
2	1	2	0	1	1	1	1	1	1	CCFS1	CCFS1		
2	1	2	3	2	2	2	1	1	1	CCFS1	CCFS1		

The following design enhancements are implemented in the code:

1. Addition of extended NSS capable in beacon, probe request/response, and association request/response.
- Enable or disable the capability through iwpriv command “ext\_nss\_cap”. Update variables in IC, which are set by default
- Function ieee80211\_add\_vhtcap ()

```
If (vap->ext_nss_cap == 1) {
    Extended_nss_capable = 1
}
```

2. Addition of supported channel width and extended NSS support field in beacon, probe request/response, and association request/response.

- Function ieee80211\_add\_vhtcap ()

```
if (cur_mode == IEEE80211_MODE_11AC_VHT80_80) {
```

```

    supported_channel_width = 0
    extended_nss_support = 1
} else if (cur_mode == IEEE80211_MODE_11AC_VHT160) {
    supported_channel_width = 0
    extended_nss_support = 2
} else if (No support for 160) {
    Supported_channel_width = 0
    Extended_nss_support = 0
}

```

3. Addition of CCFS0, CCFS1 and CCFS2 in VHT operation element and HT operation element of beacon, probe response and association response frame.

- For beacon and probe response, chan\_width is AP chan\_width and for assoc\_response, chan\_width is the negotiated chan\_width,

```

if ( chan_width == 160 || 80_80)
{
    ccfs0 = primary_channel_centre_freq
    ccfs1 = 0 or secondary_channel_centre_freq based on EXT NSS Signaling
    ccfs2 = secondary_channel_centre_freq /* present in HT operation
element based on EXT NSS Signaling */
} else {
    ccfs0 = channel_centre_freq
    ccfs1 = 0
    ccfs2 = 0
}

```

The assumption is that for QCA9984, AP NSS for 160 and 80\_80Mhz BW is always less then MAX NSS. The code must be updated appropriately if this behavior changes in HW. Revised signaling is assumed to be enabled.

4. OMN

Addition of 1-bit field in structure ieee80211\_ie\_op\_mode to indicate switching to/from 160 or 80\_80Mhz. No change in behavior that is used to indicate the new BW to FW.

### 6.12.1 Configuration commands for extended NSS signaling

- iwpriv wifiX ext\_nss <1/0>—Enables/disables extended NSS-signaling capability. EXT NSS signaling support becomes disabled if driver is not extended NSS-signaling capable.
- iwpriv wifiX g\_ext\_nss—Get extended NSS-signaling capability; 1 if enabled, 0 if disabled.
- iwpriv athX ext\_nss\_sup <1/0>—Enables/disables extended NSS-signaling support.
- iwpriv athX g\_ext\_nss\_sup—Get extended NSS-signaling support; 1 if enabled, 0 if disabled.

## 6.12.2 Limitations for extended NSS signaling

The following restrictions apply for extended NSS signaling:

- Wave2 certified wifi STAs will not be able to associate in 160MHZ with EXT NSS Signaling capable APUT and will fall back to 80MHz operation.
- EXT NSS enabled STA in 160MHz will fall back to 80MHz if legacy 160MHz AP is operational. This scenario could be optimized to use 160MHz, if STA suppresses Ext NSS signaling and continues to communicate with prop IE in association request

## 6.12.3 Sample configuration scenario for extended NSS signaling

Consider the following sample connection scenario:

1. EXT NSS AP and STA in 80, 160, 80+80MHz
2. Ext NSS AP and legacy STA in 80, 160 and 80+80MHz
3. Legacy AP and EXT NSS STA in 80, 160 and 80+80MHz

Currently, there are no clients available in market which provide the setting described in table 9.250 and table 9.75. Therefore, to verify the behavior of AP when clients advertise the settings in the table, a STA stub is used:

- The STA stub code must be able to send different NSS values for 80, 160 and 80\_80 Mhz through exttool
- Support to send OMN action frame also to be added in exttool
- Record the values populated in wmi\_peer\_assoc\_complete\_cmd in host before sending WMI to FW

**Command usage:** exttool –extnss\_stub --interface <wifiX> --extnss\_stub\_chwidth <val> --extnss\_stub\_support <val>

User must provide absolute NSS values for the 3 BW categories. Host driver stub code to do the corresponding mapping and update supported channel width and extended NSS support fields.

**Command usage:** exttool –opmode\_extnss\_stub –interface <wifiX> --opmode\_stub\_chwidth <val> --opmode\_stub\_nss <val>

User must provide the BW to which switching must occur.

Refer to the IEEE Draft P802.11REVmc\_D6.0 for further information.

Consider a sample network topology to advertise EXT NSS signaling for 160MHz on AP under test (APUT). Configure APUT and STA to enable 160 MHz bandwidth. The “VHT Extended NSS BW Capable” Bit in the “Supported VHT-MCS and NSS Set” field of the VHT Capabilities element in the Beacon and Probe Response frame are equal to one, and the Supported Channel Width Set subfield and Extended NSS BW Support subfield of the VHT Capabilities element in Beacon and Probe Response frames are set to 0 and 1 respectively. Channel Center Frequency Segment 1 in the VHT Operation element is set to 0. Channel Center Frequency Segment 2 in the HT Operation element is set to 50.

Consider a sample network topology that contains an EXT NSS APUT with legacy STA in 160 MHz with EXT NSS disabled. Configure APUT and STA to enable 160 MHz bandwidth.

Associate STA with APUT. Issue iwpriv athX ext\_nss 0. Issue iwpriv athX g\_ext\_nss. This should return 0. The “VHT Extended NSS BW Capable” Bit in the “Supported VHT-MCS and NSS Set” field of the VHT Capabilities element in the Beacon and Probe Response frame are equal to zero. STA becomes associated with AP in 160MHz.

Consider a sample network topology that contains an EXT NSS APUT with legacy STA in 80+80 MHz with EXT NSS disabled. Configure APUT and STA to enable 80+80 MHz bandwidth.

Associate STA with APUT. Issue iwpriv athX ext\_nss 0. Issue iwpriv athX g\_ext\_nss. This should return 0. The “VHT Extended NSS BW Capable” Bit in the “Supported VHT-MCS and NSS Set” field of the VHT Capabilities element in the Beacon and Probe Response frame are equal to zero. STA should get associated with AP in 80+80 MHz.

Consider a sample network topology that contains chainmask configuration. Configure APUT and STA to enable 160/80+80MHz. Associate STA with APUT. Issue iwpriv wifiX txchainmask <value>. iwpriv wifiX rxchainmask <value>. Initiate ping. Packets must be seen being transmitted with newly derived NSS based on chainmask.

## 6.13 Full-offload configuration for QCA955x

**NOTE** The full-offload is a patch release. The original tar ball (fulloffload-driver-2.9.2.069.tar.bz2) made from the 10.2.2.U2 release should be placed in the “<qsdk>/dl” directory before the build. The patches are part of the WLAN driver sources (“<wlan driver>/patches/fulloffload”), which are applied during the build process.

**NOTE** For full-offload setup, the host and target boards uses the same flash instructions as “Premium\_Beeline”. In addition to flashing the image on the host, the “Full-offload” driver host modules also need to be installed after kernel boot-up using the following commands. See the *Customize full-offload configuration for peak KPI* section for instructions on how to bring-up the full-offload driver.

```
# opkg install kmod-qca-wifi-fulloffload-host_3.3.8+2.9.2.069-1_
ar71xx.ipk
```

Full-offload support is not available in QCA\_Networking\_2017.SP.F.5.x.

### 6.13.1 Flash boot mode

Full offload mode is verified using the AP136-21x or AP135-22x host platform and AP135 20x + CUS 223 as target platform.

- Both the AP136/AP135 host and AP135 target are booting from independent flashes connected over CAT5 cable.
- AR8327 switch on the AP136/AP135 host platform is used forming switch in the middle configuration between host and target.
- All radio/data flow tests are performed on the AP136/AP135+AP135 board combination.

- The following diagram shows the host-target connectivity over the CAT5 cable. The LAN Port4 of the AP136/AP135 host must be connected to WAN port of AP135.

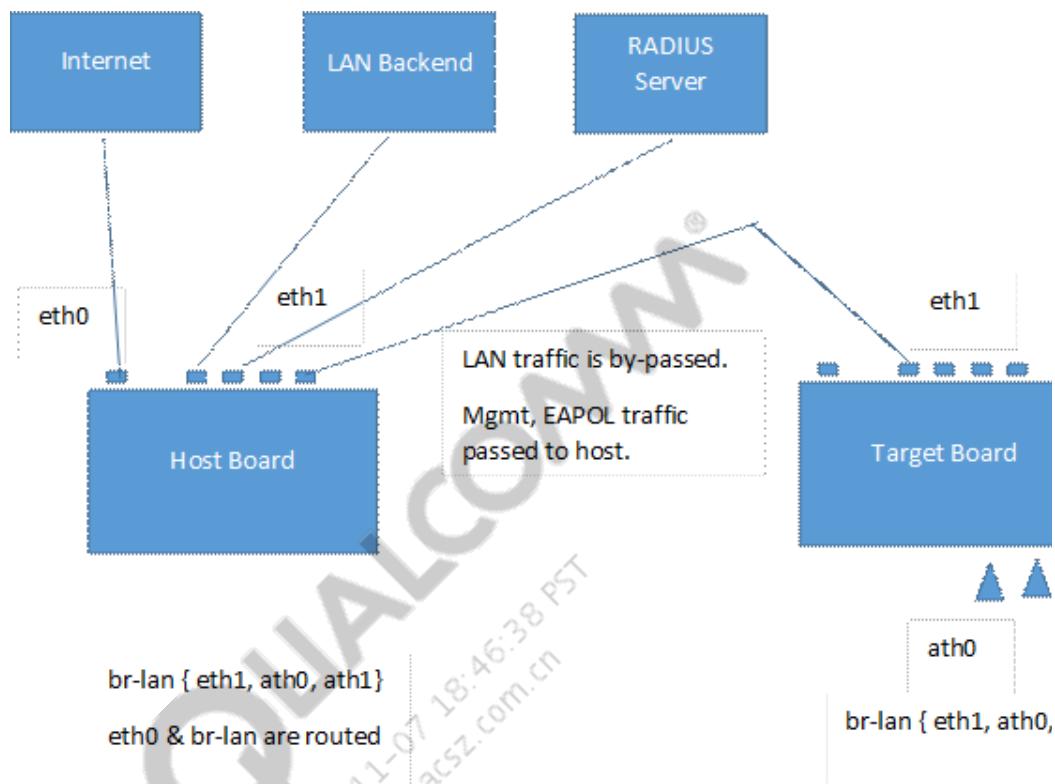


Figure 6-16 Host-target connectivity over CAT5 cable

### 6.13.2 MII boot

MII boot for AP135/CUS223 offload is verified using (host: AP136-22x) + (target: AP135-20 + CUS 223) combination connected over the bridge card.

- MAC-to-MAC combination verifies the boot over RGMII connectivity between the two.
- This platform combination primarily showcases the first stage firmware download over MDIO and second stage download over RGMII.

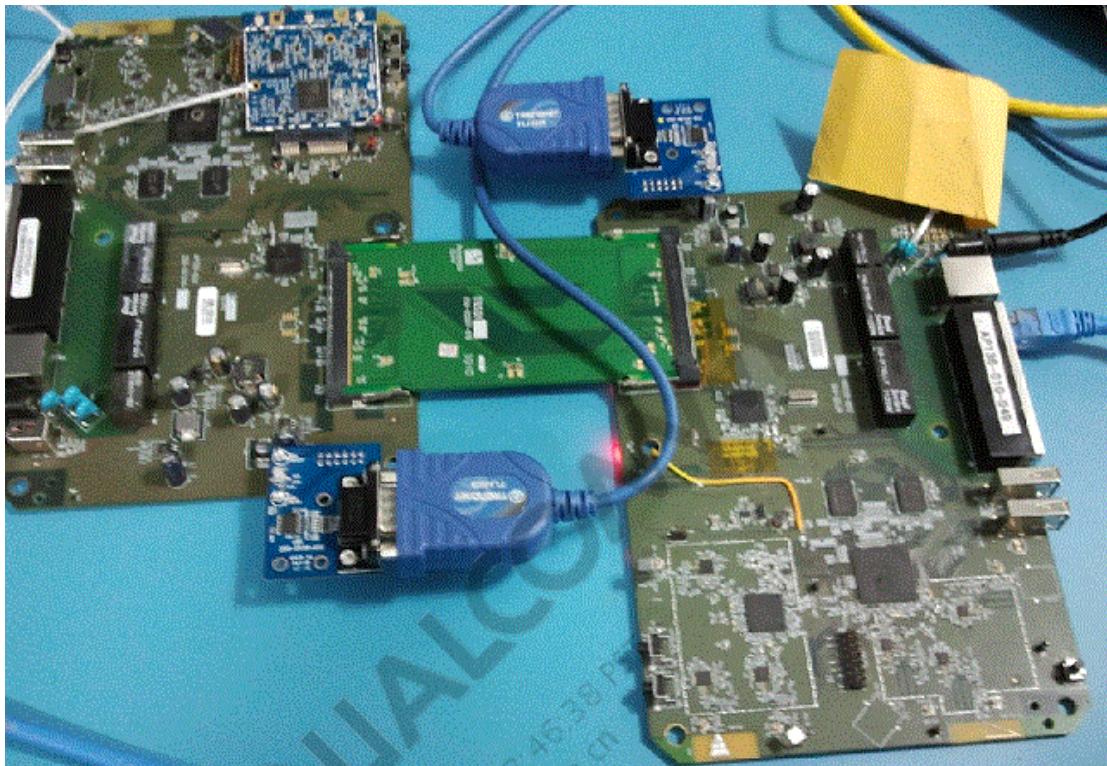
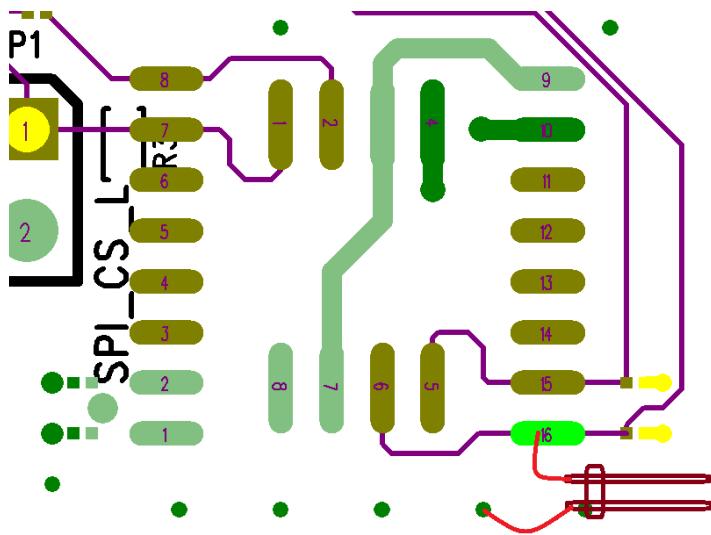


Figure 6-17 MII boot

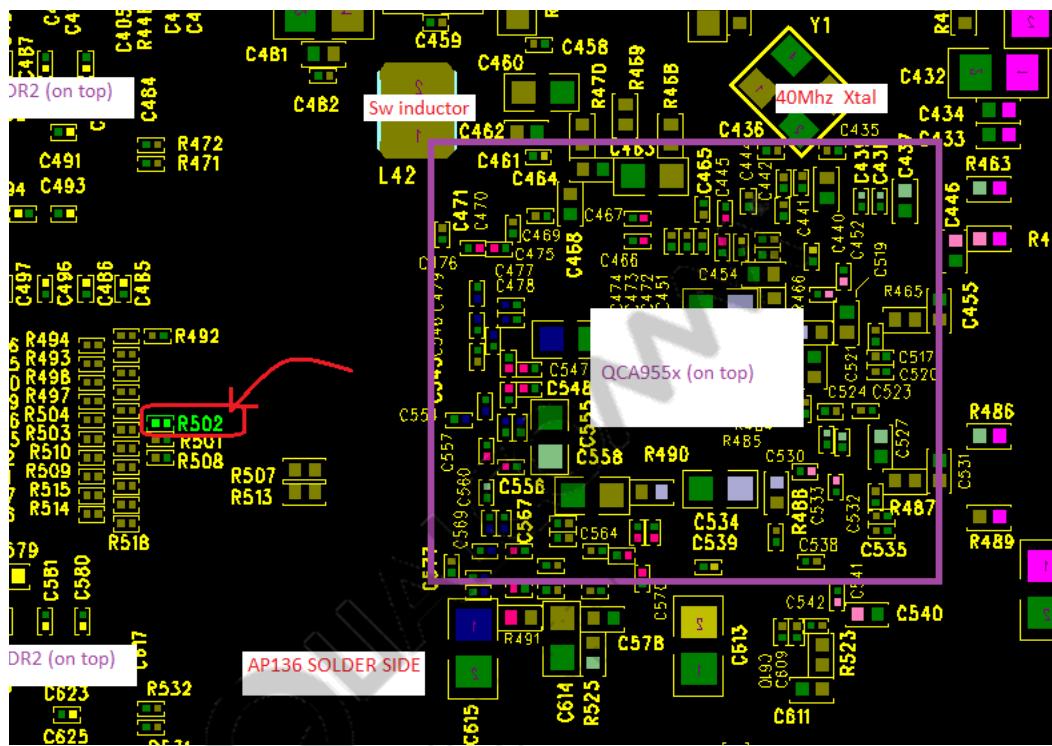
#### 6.13.2.1 Reworking the target AP135-0xx for MII Bootrom mode

To permanently set into Bootrom mode:

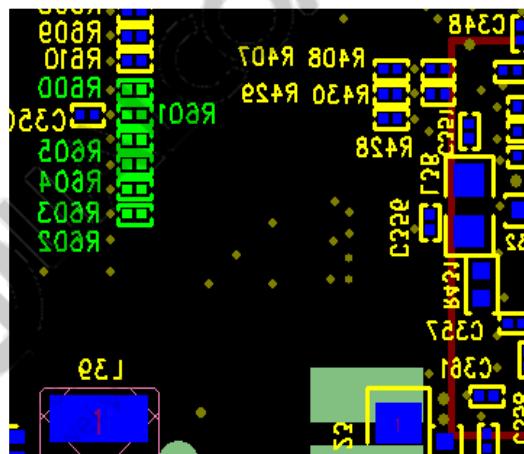
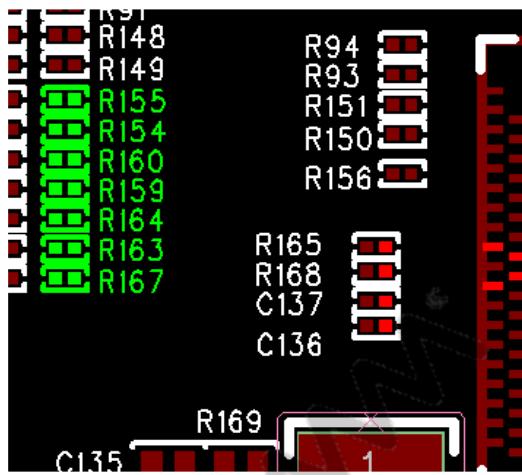
1. De-stuff R390 (solder side next to pin 16 of flash U5):



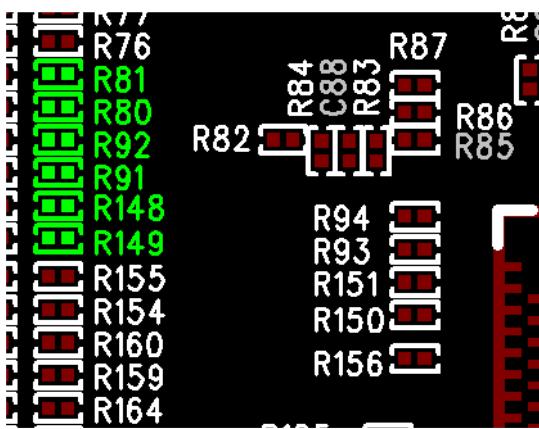
2. Set for Bootrom MDIO mode (DDR\_ADDR4 = 10K to GND) using  
Load R502 =10K:

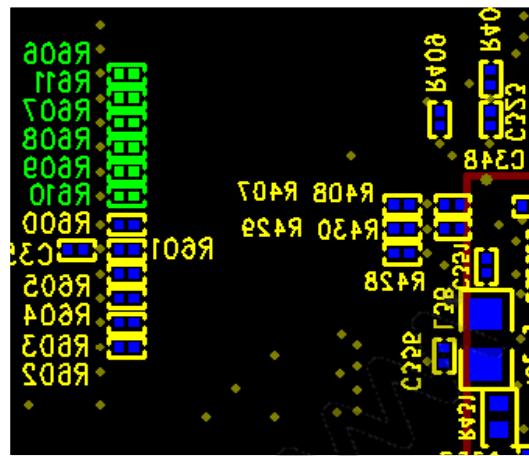


3. De-stuff pullup on GPIO18 (10K) to avoid double pullup on MDIO line from host (destuff R528).
  4. Remove R167 to disconnect S17\_INT from GPIO11 (top side left to AR8327N).
  5. Remove R154, R163, R164, R159, R160, R155, R600, R601, R602, R603, R604, and R605 to disconnect the AR8327N RGMII\_TX with QCA955x:

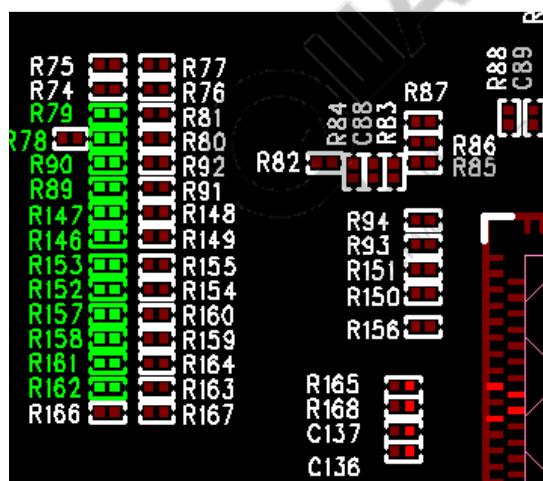


6. Remove R81, R80, R92, R91, R148, R149, R606, R611, R607, R608, R609, and R610 to disconnect the AR8327N RGMII\_RX from QCA955x:

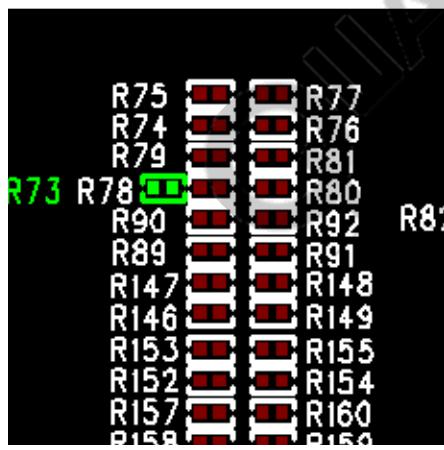
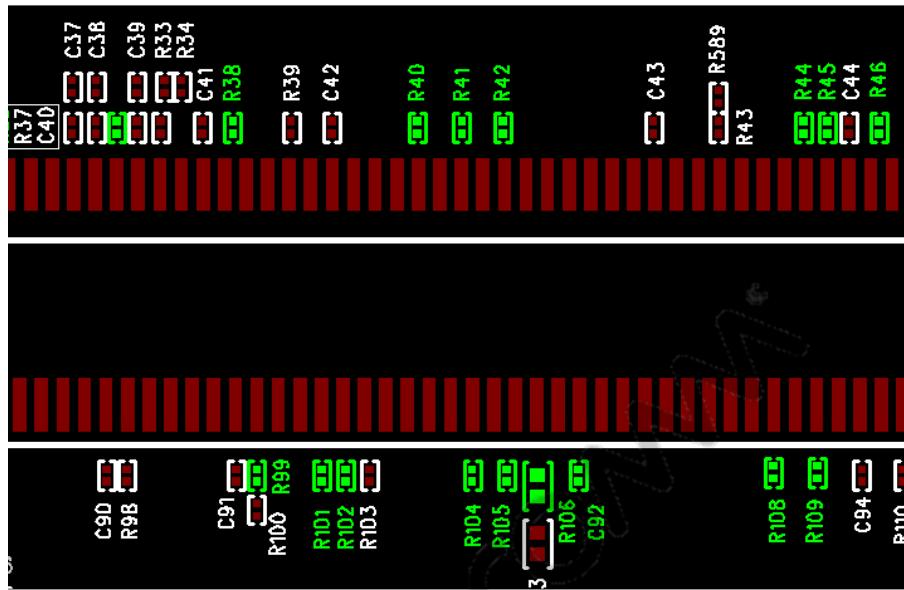




7. Mount R79, R78, R90, R89, R147, and R146 to connect XMII\_RX with QCA955x.
  8. Mount R153, R152, R157, R158, R161, and R162 to connect XMII\_TX with QCA955x.
  9. Mount R166 and R100 to connect GPIO11 and GPIO18 with XMII connector:



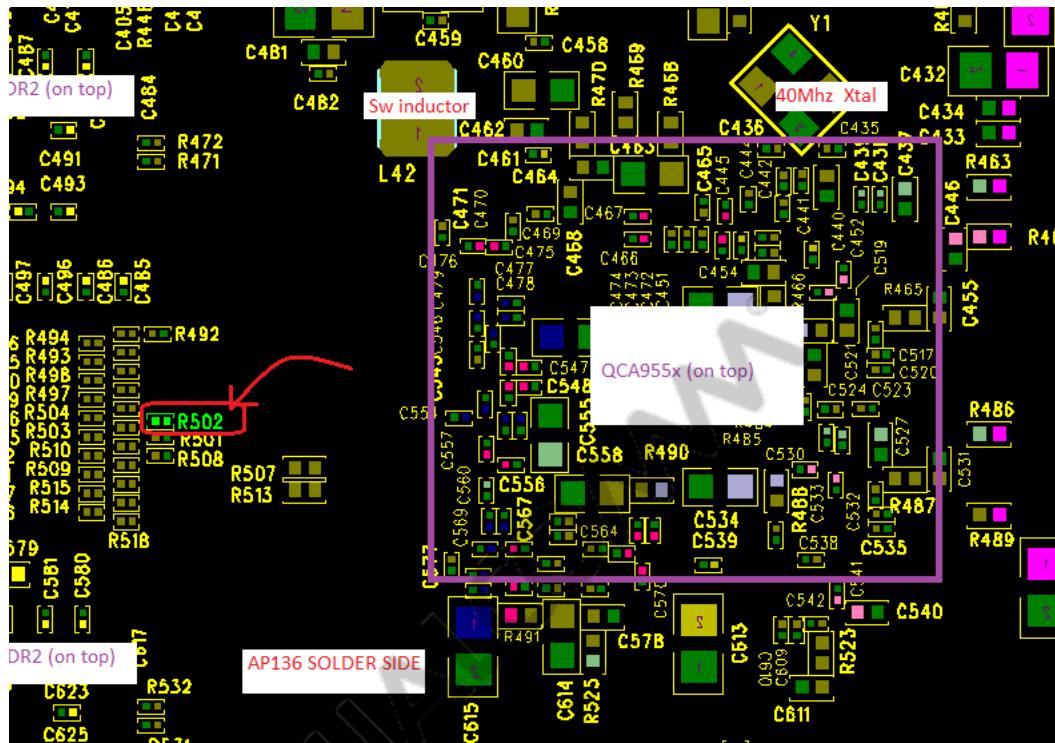
10. Mount R36, R46, R45, R109, R44, R108, R73, R105, R42, R104, R41, R40, R101, R102, R106, R38, R99 to connect XMII signals with XMII connector.



### 6.13.3 Reworks for host AP136-0xx for host mode

**NOTE** Ensure R390 is loaded (10K). The host board boots out of its own flash.

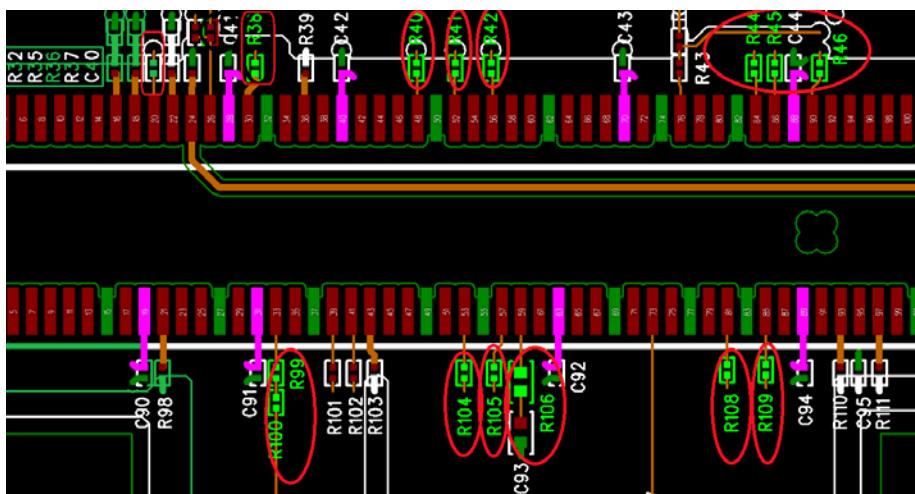
1. R74, R75, R101, and R102 are to be loaded with  $0\ \Omega$  (connect EMDIO, EMDC to xMII connector pin 41, 39).
2. To set for bootrom MDIO mode (DDR\_ADDR4 = 10K to GND),  
Load R502 = 10K:



3. De-stuff pullup on GPIO18 (10K) to avoid double pullup on MDIO line from host: Destuff R528

Reworks near the xMII connector, to connect the RGMII across to the connector.

Populate all highlighted locations with  $0\ \Omega$ : 15 places (top side near xMII connector) R36, R38, R40, R41, R42, R44, R45, R46, R99, R104, R105, R106, R108, R109, and R100.

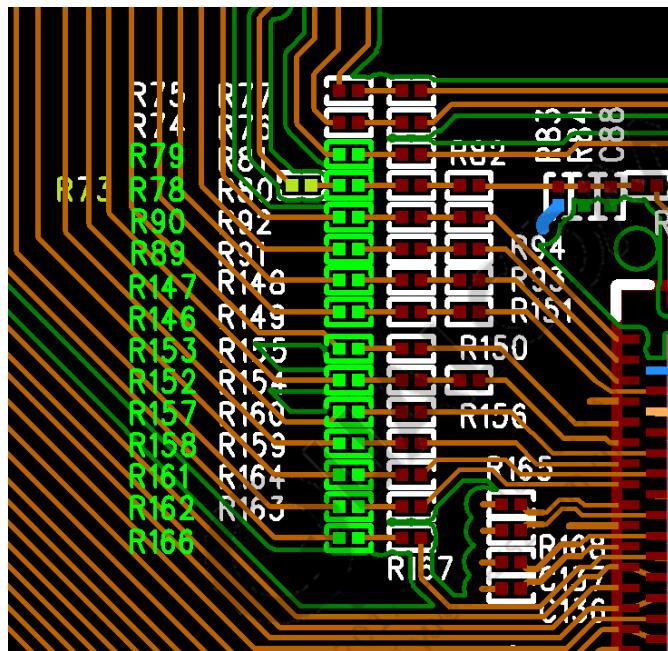


4. Rework near the AR8327N switch to disconnect the RGMII connections from the AR8327N and connect it towards the xMII connector.

5. Populate the highlighted resistor locations in the figure below with  $0\ \Omega$ ; de-stuff the corresponding resistor on the right if stuffed. These are located close to the AR8327N switch on top side of the board.

Locations to stuff with  $0\ \Omega$ : R73, R78, R79, R90, R89, R147, R146, R153, R152, R157, R158, R161, R162, and R166 (14 res)

Locations to de-stuff if stuffed: R81, R80, R92, R91, R148, R149, R155, R154, R160, R159, R164, R163, and R167 (13 nos).



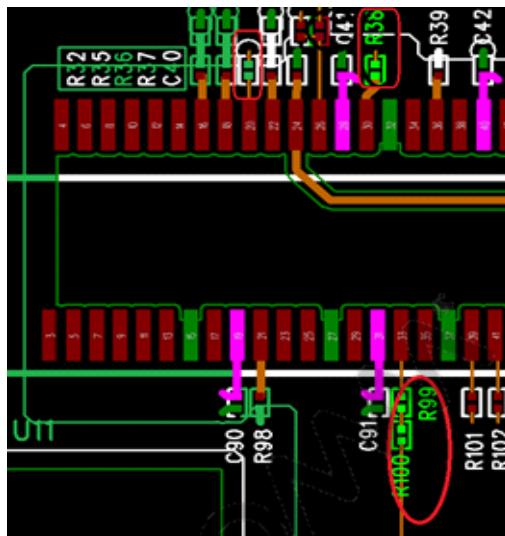
**NOTE** At this point the board is connected with the RGMII port of the QCA955x connected to the xMII connector. Use a TB681 to connect to another AP136 (target board).

#### 6.13.3.1 Reworking the target AP135-0xx for host mode

**NOTE** Ensure that MDIO and MDC are connected across the xMII connector at their usual place (pins 39 and 41). The interconnect boards will route these signals to pins 30 and 33 respectively on the XMII connector of the target board.

Ensure R74, R75, R101 and R102 are stuffed on the host board with  $0\ \Omega$ .





## 6.14 Save system memory content for troubleshooting NSS problems

The mdump utility enables system memory content to be saved for monitoring and debugging NSS firmware (FW) problems after a system crashes. By default, thus utility records 16 MB NSS FW DDR content, starting from address 0x4000\_0000. The 16 MB of data is the minimum DDR information required for debugging NSS FW crash. If a larger amount of memory content can be saved or dumped, examining and diagnosing failures regarding the system-crash becomes easier. However, the maximum output file size is 512 MB. If the system memory is larger than 512 MB, then only up to 512 MB of data can be recorded at a time. If only one DDR file is dumped, it is named as NSSDDR.BIN. If multiple DDR files are created, they are named as EBICS0.BIN, EBICS1.BIN, EBICS2.BIN, and so on.

The mdump tool has two options: `-a #####` and `-b #####`, where ##### can be either a hexadecimal or digital number that specifies the dump size or dump start-address. The `-b bytes` option enables changing the dump size, and the `-a address` option enables changing the dumping start address in DDR.

Besides recording DDR, NSS also has a tight-couple memory (TCM) memory that is necessary to be saved for troubleshooting purposes. The TCM starting address varies across chipsets. For the IPQ806x chipset, the TCM starting address is at 0x3900\_0000. The TCM size is 128 KB (up to 256KB) and command for dumping TCM for IPQ806x chipsets is as follows:

```
mdump -a 0x39000000 -b 131072 > /usb/NSSTCM.BIN
```

Therefore, at least two memory regions must be dumped for NSS FW debugging.

When each subsystem crashes, the corresponding subsystem generates crash dump in DDR and triggers a system-panic for reboot. To prevent panic from occurring during NSS FW crash, enter the following command after system boots up to collect crash dump (core dump):

```
echo -1 > /proc/sys/dev/nss/general/coredump
```

This command causes the NSS subsystem to not panic after generating a core dump. Instead, NSS core dump process displays the following message on the console:

```
NSS core 0 signal COREDUMP COMPLETE
NSS core 1 signal COREDUMP COMPLETE
NSS core dump completed and please use mdump to collect dum data
```

When this message is displayed, plug in a USB memory stick and use mdump to record the system memory data. The recorded data can be analyzed for rectifying faults.

## 6.15 Known networking acceleration limitations

- Fragmentation does not occur on frame sizes of 2049 bytes with default interface MTU and switch maximum frame sizes set as 9000 bytes. The fragmentation and reassembly functionality is only supported on enterprise products.
- DSCP remarking value remains the same for bidirectional flows, even when different outgoing DSCP values in iptables are configured. This NSS performance on DSCP remarking is an expected behavior.
- Serial connection is not accessible when dual-stack, bidirectional randomized packet traffic is run with SFE acceleration engine enabled. The serial and Telnet sessions are accessible only after the traffic is stopped, and not accessible when traffic is present. Console access is slow, while running 8K flows with SFE enabled on IPQ8064.
- Throughput is observed to be less by 150 Mbps - 200 Mbps when IPv4/IPv6 (dual-stack) TCP traffic is run with SFE acceleration engine enabled on an AP. KPI for SFE on IPQ8064 chipsets is lower compared with the KPI on IPQ4019 chipsets or non-SFE (ECM + NSS hardware) on IPQ8064 chipsets. Optimization is required outside of SFE.
- IPsec LAN to WAN traffic does not resume after a network restart. With the Openwrt upgrade, the firewall restarts even if it was disabled before. This happens because miniupnpd tries to reload the firewall when it finds that the firewall has not started. It is recommended to disable miniupnpd and also the firewall.
- Tunnel does not reconnect after disabling/enabling WAN interface eth0. When the physical interface (over which L2TP tunnel is established) is brought down using ifconfig command, the corresponding notification is not sent to xl2tpd, and it does not restart when interface status changes. xl2tpd assumes that the interface always exists. This is a xl2tpd design limitation.
- "rfs\_wxt\_get\_cpu\_by\_irq[131]:INFO:IRQ is bound to be more than one CPU" prints on console leads to 2.4 GHz beacon stuck. In some special conditions, a continuous "rfs\_wxt\_get\_cpu\_by\_irq[131]:INFO:IRQ is bound to more than one CPU" prints on console leading to 2.4 GHz beacon stuck. If required, this message can be turned off using: echo 1 > /proc/qrfb/debug
- L2TP session over PPPoE WAN mode cannot be started. With Openwrt upgrade, a blackhole rule gets added from /etc/ppp/ip-up.d/add-tunnel-server-blackhole: ip route add blackhole \$svr metric 1000. This blocks traffic flow. PPPoE adds blackhole to avoid route looping issue when the WAN connection is disconnected. When the PPPoE peer IP address is the L2TP server IP address, it makes the L2TP route conflict with blackhole route. It is suggested not to use the peer PPPoE IP address as the L2TP server IP address.

- Untagged interface learns MAC address for same IP of the interface. Linux kernel function "fib\_validate\_source()" has changed in 3.14. The "/proc/sys/net/ipv4/conf/br-lan/rp\_filter" parameter is added to check if source IP must be validated while looking up the route. Its default value is false, so the source IP is not validated, and a neighbor for duplicated IP address is created. Linux kernel 3.4 does not learn a neighbor for local IP, and does not send an ARP response to notify IP address conflict. However, in Linux kernel 3.14, it sends out an ARP response to notify the IP address conflict. "rp\_filter" can be enabled to revert to the Linux kernel 3.4 behavior but this is not recommended. Though a neighbor for local IP is learned, this neighbor is never used and does not affect any network function.
- Multicast traffic does not run after enabling mcastenhance. Run multicast traffic from ROOTAP backbone to WDS STA backbone when multicast enhancement was set to 2. The multicast traffic is not forwarded to the wds stations. This is because the report was not forwarded when snooping is enabled. Run igmp proxy on the root AP or set the flood command on the non-ROOTAP: mcsctl -s br-lan route flood.

## 6.16 Multiple WAN

This section describes how to set up load-balancing and failover mechanism on the multiple WAN connection. The package Multiwan provides a simplified mechanism to configure the load-balancing and failover capabilities on the multiple WAN (multiwan) connection.

To start and stop the multiwan functionality, the /etc/init.d/multiwan start and /etc/init.d/multiwan stop commands are available.

### 6.16.1 UCI configuration of the multiwan

Option	Default	Value range	Description
Section : config			
Enable	0	0   1	Enable multiwan or not
default_route	balancer	balancer/fastbalancer/<interface>	Select the default route for all the unspecified traffic
debug	0	0   1	0 : disable 1:enable
health_monitor	parallel	parallel/serial	Serial can save some system resource
Section : interface			

<b>weight</b>	10	<i>disable/1-10</i>	The weight in the load balance calculation
<b>health_interval</b>	10	<i>disable/5/10/20/30/60/120</i>	Health interval in seconds
<b>icmp_hosts</b>	dns	<i>disable/dns/gateway/&lt;host&gt;</i>	The Host to be pinged
<b>timeout</b>	3	<i>disable/1-5/10</i>	Time out in the icmp
<b>health_fail_retries</b>	3	<i>1/3/5/10/15/20</i>	Attempt before the fail over
<b>health_recovery_retries</b>	5	<i>1/3/5/10/15/20</i>	Attempt before the recovery
<b>failover_to</b>	fastbalancer	<i>disable/balancer/fastbalancer/&lt;interface&gt;</i>	Traffic failover destination
<b>dns</b>	auto	<i>auto/&lt;dns&gt;</i>	The dns server's ip address, auto will use the dns get from the wan connection

## Section : mwanfw

<b>src</b>	All	<i>all/&lt;IP&gt;/&lt;hostname&gt;</i>	Src ip address
<b>dst</b>	all	<i>all/&lt;IP&gt;/&lt;hostname&gt;</i>	Destination ip address
<b>port_type</b>	dports	<i>dports/source-ports</i>	Dst port or src port
<b>proto</b>	all	<i>all/tcp/udp/icmp/&lt;custom&gt;</i>	Mapped port number in the switch
<b>ports</b>	all	<i>all/&lt;port,port:range&gt;</i>	Ports or port range
<b>wanrule</b>		<i>all/tcp/udp/icmp/&lt;custom&gt;</i>	Wan uplink
<b>failover_to</b>		<i>balancer/fastbalancer/&lt;interface&gt;</i>	Destination when fail over

## 6.16.2 Sample multiwan connection

Consider a sample scenario in which two PPPoE WAN connections on the PPP server side are created. These connections are used to create two accounts for users, namely isp1 and isp2. Also , the multiwan configuration is created in this example.

To create the WAN connection, enter the following commands:

```
uci set network.wan=interface
uci set network.wan.proto=pppoe
uci set network.wan.ifname=eth0
uci set network.wan.ac=isp1
uci set network.wan.username=bob
uci set network.wan.password=test
uci set network.wan2=interface
uci set network.wan2.proto=pppoe
uci set network.wan2.ifname=eth0
uci set network.wan2.ac=isp2
uci set network.wan2.username=mary
uci set network.wan2.password=test
uci commit
/etc/init.d/network restart
```

To configure firewall, enter the following commands:

```
uci set firewall.@zone[1].network=
uci add_list firewall.@zone[1].network=wan
uci add_list firewall.@zone[1].network=wan2
uci add_list firewall.@zone[1].network=wan6
uci commit
/etc/init.d/firewall restart
```

To configure multiwan, enter the following commands:

```
uci set multiwan.wan=interface
uci set multiwan.wan.weight=3
uci set multiwan.wan.health_interval=4
uci set multiwan.wan.icmp_hosts=gateway
uci set multiwan.wan.timeout=3
uci set multiwan.wan.health_fail_retries=3
uci set multiwan.wan.health_recovery_retries=5
uci set multiwan.wan.failover_to=balancer
uci set multiwan.wan.dns=auto

uci set multiwan.wan2=interface
uci set multiwan.wan2.weight=3
uci set multiwan.wan2.health_interval=4
uci set multiwan.wan2.icmp_hosts=gateway
uci set multiwan.wan2.timeout=3
uci set multiwan.wan2.health_fail_retries=3
uci set multiwan.wan2.health_recovery_retries=5
uci set multiwan.wan2.failover_to=balancer
uci set multiwan.wan2.dns=auto
uci commit multiwan
/etc/init.d/multiwan restart
```

To set up customized rules, enter the following commands:

```
uci add multiwan mwanfw
uci set multiwan.@mwanfw[-1].src=192.168.1.0/24
uci set multiwan.@mwanfw[-1].dst=www.google.com
uci set multiwan.@mwanfw[-1].wanrule=balancer
uci commit multiwan
/etc/init.d/multiwan restart
```

### 6.16.3 Analyze the multiwan

The following commands are used to analyze the multiwan function:

To list the route table:

```
cat /etc/iproute2/rt_table
#
# reserved values
#
255    local
254    main
253    default
0      unspec
#
# local
#
#1     inr.ruhep
#
170 LoadBalancer
171 MWAN1
172 MWAN2
```

To view IP rules:

```
ip rule show
0:      from all lookup local
9:      from all fwmark 0x1 lookup LoadBalancer
10:     from 192.168.10.30 lookup MWAN1
11:     from all fwmark 0x10 lookup MWAN1
20:     from 192.168.20.30 lookup MWAN2
21:     from all fwmark 0x20 lookup MWAN2
32766:  from all lookup main
32767:  from all lookup default
```

To view IP route tables:

```
ip route show table LoadBalancer
default proto static
    nexthop via 192.168.10.1 dev pppoe-wan weight 10
    nexthop via 192.168.20.1 dev pppoe-wan2 weight 10
192.168.1.0/24 dev br-lan proto kernel scope link src 192.168.1.1
192.168.10.1 dev pppoe-wan proto kernel scope link src 192.168.10.30
192.168.20.1 dev pppoe-wan2 proto kernel scope link src 192.168.20.30
```

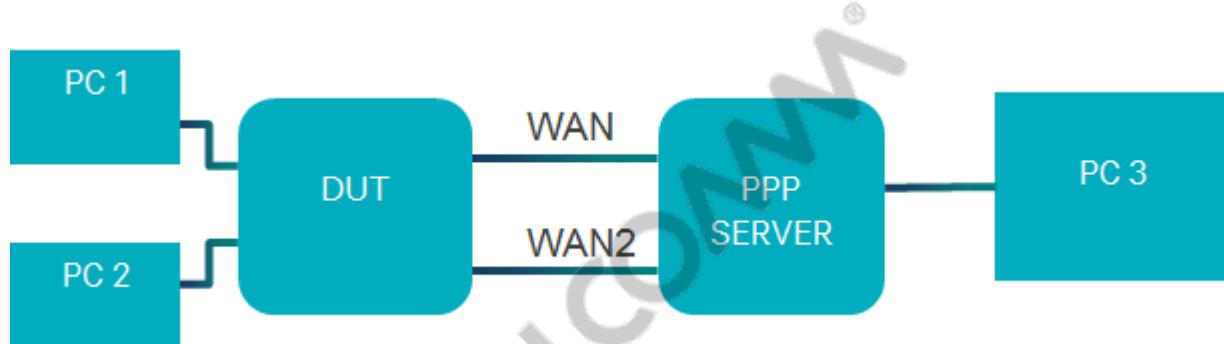
To view the filters in the iptable:

All the related filter was inserted to the mangle chain, it is possible to view all the filter by the command:

```
iptable -nvL -t mangle
```

#### 6.16.4 Sample multiwan tests

For load-balancing test, consider the following topology:



Assuming that both of the load balance weight are 10, run 20 flows to the destination behind the wan side

Expected behavior:

10 flows are on the WAN

10 flows are on the WAN2

Because the kernel scheduling of the load balancing is based on the random.

It is not one by one on the different flow, but it will be equal to the wight sum of the setting.

The extremely result of the first 10 flows go through the WAN connection, the 10 later flows go through the WAN2 connection.

For failover test using the topology, tear down WAN connection. All the traffic will go through the WAN2 connection.

#### 6.16.5 Limitations

In the implementation, the lowest 8 bits in the skb mark is used; therefore, a conflict with StreamBoot, is observed because it uses all of the Mark field. This implementation use iptable to filter and mark the stream, so the firewall cannot be disabled

Load balancing of the traffic occurs based on the flow, it load balance regardless of how much traffic on the flow. The conntrack restore and the cache of the route are used. Therefore, the route cached continues to be used if there is no route change in the system and the route is not modified thereafter.

## 6.17 Multipath support in ECM

This section describes how the multipath support is added to ECM and the components of the ECM that are impacted.

### 6.17.1 Single path implementation

ECM is the central connection manager which is hooked into the netfilter post routing and bridge post routing points. It monitors the flows and extracted the information from the flows to create a connection in its database and pushes the acceleration rule to underlying acceleration engine (NSS or SFE).

Some of the information that it needs to create the connection are the system's net device interfaces on which the connections are flowing through and the MAC addresses of the nodes which are communicating with each other.

It is doing route lookups by using the nodes' IP addresses to extract these information.

ECM was designed only considering the single path which means if there is only a single default route in the system. When we introduce the multipath which means the existence of multiple default routes in the system, the route lookup based on the IP addresses does not work. Each route lookup causes to create arbitrary route entries in the route cache table.

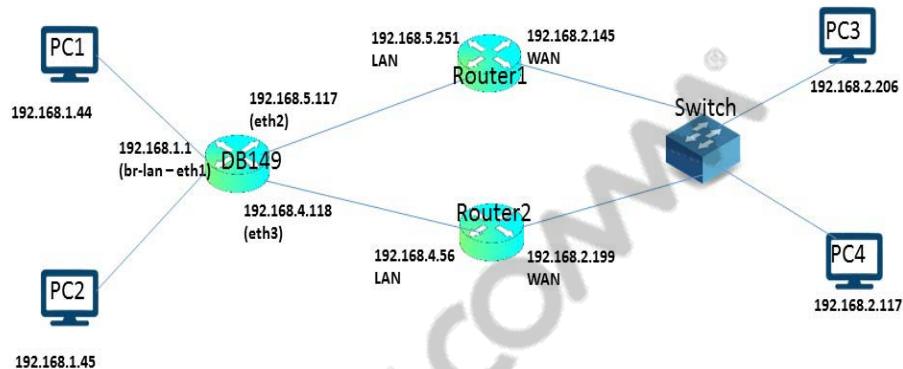
### 6.17.2 Multipath model

For the routed traffic, the skb has a route table field which has the route information for that packet. That information contains the source and destination interface of the packet, source, destination and gateway IP addresses of the packet. It is not necessary to perform route lookup to find the netdev interface instances. Use the skb to construct the ECM's interface hierarchies for source and destination sides (between the two sides).

After deciding the source and destination interfaces, the node MAC address lookups can also be done in the neighbor entries of that net device. This eliminates another route lookup in the mac address lookup functions.

For the bridging traffic, no problems occur because the skb's input/output interfaces are already used, and also the Ethernet header's source and destination MAC addresses are used for the nodes. This information from the skb is used for bridging traffic.

### 6.17.3 Multipath Test Topology



```
ip route replace default nexthop via 192.168.5.251 dev eth2 weight 1
nexthop via 192.168.4.56 dev eth3 weight 1
```

As illustrated in the preceding figure, DB149 is the device under test (DUT), which runs with the ECM. The Router1 and Router2 can be any commercial router.

The goal is to send 2 separate flows from PC1 to PC3 and PC2 to PC4 and distribute these flows through the WAN1 and WAN2 interfaces of the DUT. The DUT will have 2 default routes with specific weight values

The following is the command to create these default routes in the system.

```
ip route replace default nexthop via 192.168.5.251 dev eth2 weight 1
nexthop via 192.168.4.56 dev eth3 weight 1
```

After running this command, the 2 traffic should go through the separate WAN interfaces of the DUT. This can be monitored on the PC3 and PC4 with wireshark. On the wireshark captures of PC3 and PC4, the source IP addresses of the packets should be different than each other. If it is 192.168.2.145 on the PC3, it must be 192.168.2.199 on PC4.

## 6.18 IPv6 MAP-T

*Mapping of Address and Port using Translation (MAP-T)* implements ipv4 packets to be tunneled in service providers' ipv6 network without tunnel header overhead. This is stateless implementation so service providers need not maintain any states with respect to flow. All needed information for v4 to v6 and vice versa are encoded in ipv6 address of the packet itself. Statefulness is maintained at CE for NAT44 translation; but NAT46 is still stateless.

The packet oriented data network discussed in this chapter is one that is implemented over Internet Protocol (IP). This chapter discusses NSS acceleration support for MAP-T UDP and TCP packets. There is no control packets between BR and CE. Packet acceleration support is only at CE. Packet acceleration at BR is not implemented.

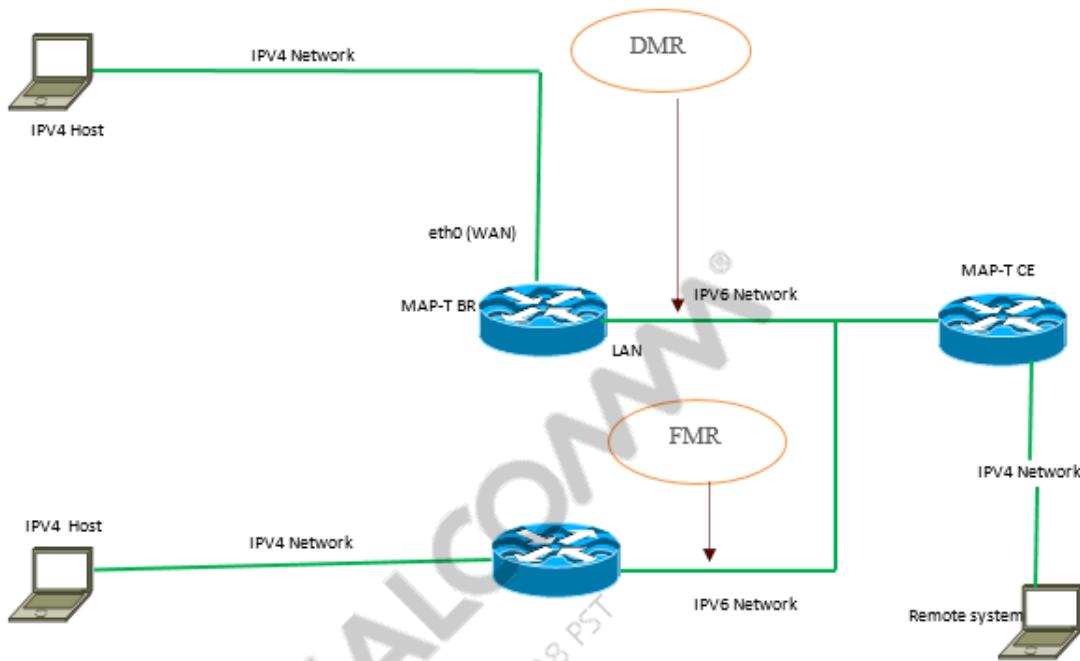
NAT-46 (external feeds that implement MAP-T in slow path) supports multiple styles – map-t , rfc6052 , and map-0. But map-t rfc (RFC 7599) states that map-t local style is only map-t and remote style can be either map-t (FMR rule) or RFC 6052(DMR rule). Acceleration engine may or may not accelerate other combinations (of styles) but no guarantee.

The following table defines the terms and acronyms used in the discussion of this chapter:

NSS	Networking Sub System
HLOS	High Level Operating System
ECM	Enhanced Connection Manager
IP	Internet Protocol
CME	Connection Match Entry
CE	Customer Equipment which implements a MAP-T
BR	MAP Border Relay

### 6.18.1 MAP-T topology

CE should implement a MAP-T functionality and should be able to communicate with any other Node in map domain or outside map domain slow path. UDP and TCP flows (which are already established in slow path) can be offloaded to acceleration engine.



This feature accelerates all MAP-T tcp/udp flows. NAT44 can translate only ported flows (which is UDP, TCP and ICMP); therefore, MAP-T works only on those. IPv6 flows cannot be carried over MAP-T.

### 6.18.2 Design changes

#### ECM

The following updates to ECM are implemented:

- ecm\_interface\_hierarchy\_construct() to be modified to process MAP-T interface. Create hierarchy with only one interface. Sub interface is not needed to be added into the hierarchy.
- Support to be added to output of ecm\_dump.sh to show MAP-T-related information.
- MAP-T uses nat64 kernel module, which comes as an external feeds to
- OpenWrt. Patching this module is done to add API, which can be used by ECM or client code.
- Xlated src address is not a local address. We need to modify ecm\_interface\_mac\_addr\_get().
- ECM should must acceleration rules to NSS.

#### NSS client

Although MAP-t rules can be configured after the interface is brought up, a restriction will be put forward to disallow this practice. All rules should be configure before the interface is brought up.

Nat46 external package should be modified to keep the list of map-t devices created and should provide an EXPORTed function by which any other entry can verify whether particular netdev is of type MAP-T.

There should be only one map-t netdevice per DUT for map-t client function to work properly. MAP-T documents available talks about configuring default ipv4 route as map-t device. It is possible to subnet and have multiple map-t device because as nat46.ko supports. Therefore, these multiple map devices are supported.

The NSS MAP-T client registers to listen for the following netdevice notifiers:

**NETDEV UP:** As mentioned previously, rules are configured prior to bringing up the interface. NETDEV UP call back can call package function (mentioned in preceding paragraph) to identify map-t netdevice and access the map-t rules. This routine must call nss driver API to create an nss dynamic interface and configure it with parameters which are extracted from map-t rules. This interface is of type ARP\_HDR\_NONE.

**NETDEV\_DOWN:** This notifier callback sends an interface down message to the MAP-T package of NSS firmware. It should delete the dynamic interface.

NSS statistics must be synced to host and same should be reflected in interface stats. NSS debug stats are implemented in debugfs.

In the NSS Exception Path, packets can be exempted after

1. Packets are exempted before v4 → v6 xlate: (LAN to WAN)  
Packet is transmitted through map-t interface
2. Packets Translated to IPv6 (LAN to WAN). Exception as there is no ipv6 rule.  
Strip off ipv6 header, add ipv4 header and transmit through map-t interface.
3. Packets exception before v6→v4 xlate (WAN to LAN)  
Packet is transmitted through map-t interface
4. Translated to IPv4 (WAN to LAN). Exception as there is no ipv4 rule.
5. Strip off ipv4 header, add ipv6 header and xmit through map-t interface

The following updates are implemented for NSS driver:

1. Export headers that define the interface between host and MAP-T package within NSS. This includes:
  - a. Messages sent from host to NSS
  - b. Events received from NSS to host
2. Export APIs that can be used by other modules for the following:
  - a. Send MAP-T messages to NSS
  - b. Receive MAP-T related events and callbacks
  - c. Register/unregister MAP-T interfaces from NSS
3. New dynamic interface type will be added to support MAP-T

## References

- RFC 7599 MAP-T RFC
- RFC 6145 IP\_ICMP Translation Algorithm

## 6.19 IPv6 NAPT

The IPv6 feature in the NSS DUT is supported. IPv6 is an addressing mechanism suggested by IETF to take care of the dwindling number of public IPv4 addresses available for users across the world. IPv6 address is 16 bytes (as compared to 4 bytes used for IPv4 address) and is expected to take care of future unique Layer 3 address requirements during the move towards an all IP digital world.

Initial adoption of IPv6 addresses was slow due to various reasons (including cost of replacement of existing IPv4 infrastructure and training of system administrators on new IPv6 infrastructure). NAPT (Network Address and Port Translation) helped push the exhaustion of all IPv4 addresses by a few years.

The principles that must be followed and the needed for an IPv6 header is similar to that for an IPv4 header. However, there are some major differences in header fields. For example, a IPv6 header follows a next header convention for extra options that is different from the increase in the header size convention followed for an IPv4 header. There is no header checksum and header length field in IPv6 header.

Neighbor discovery and DHCP protocols are slightly different for IPv6. Also, NAPT is not required for IPv6 as there are enough globally unique IPv6 addresses available. Native Dual Stack is the ideal solution for IPv4 to IPv6 transition, provided enough IPv4 addresses are available. 6RD (IPv6 Rapid Development) provides IPv6 network services over an IPv4 network using a tunnel to the 6RD border relay gateways. 6RD border gateways use IPv4 any cast addresses for failover/resiliency. Dual Stack Lite is used to provide IPv4 services to host behind the IPv6 ISP network and IPv6 addressable gateway. It uses the IPv4 in IPv6 tunnel. MAP-T is used to carry IPv4 traffic over IPv6 ISP network by translating IPv4 address in to IPv6 address.

### 6.19.1 Detailed information on tunneling with IPv6

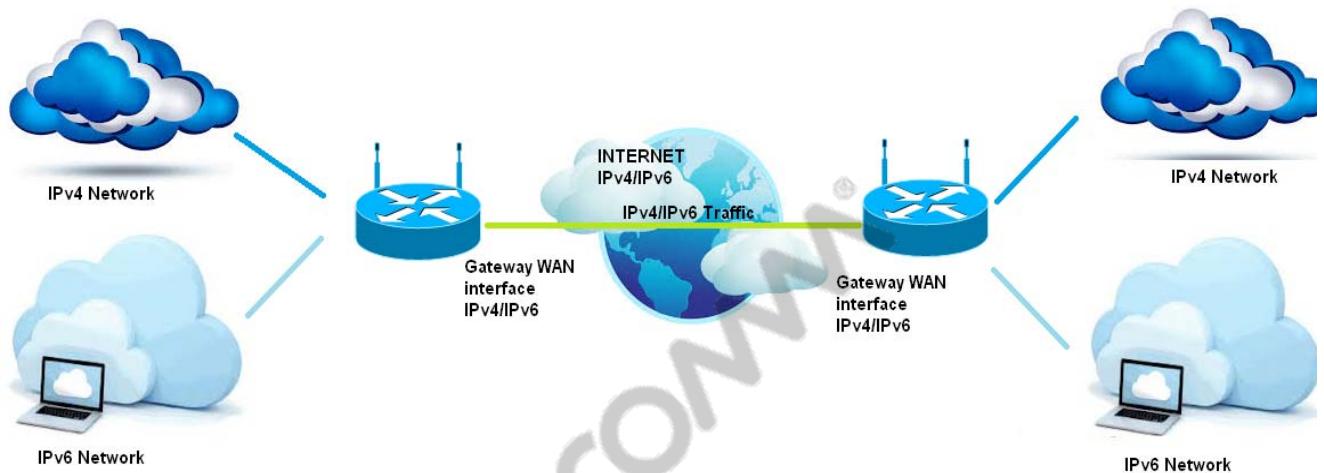


Figure 6-18 Native Dual-Stack Co-existence Mechanism

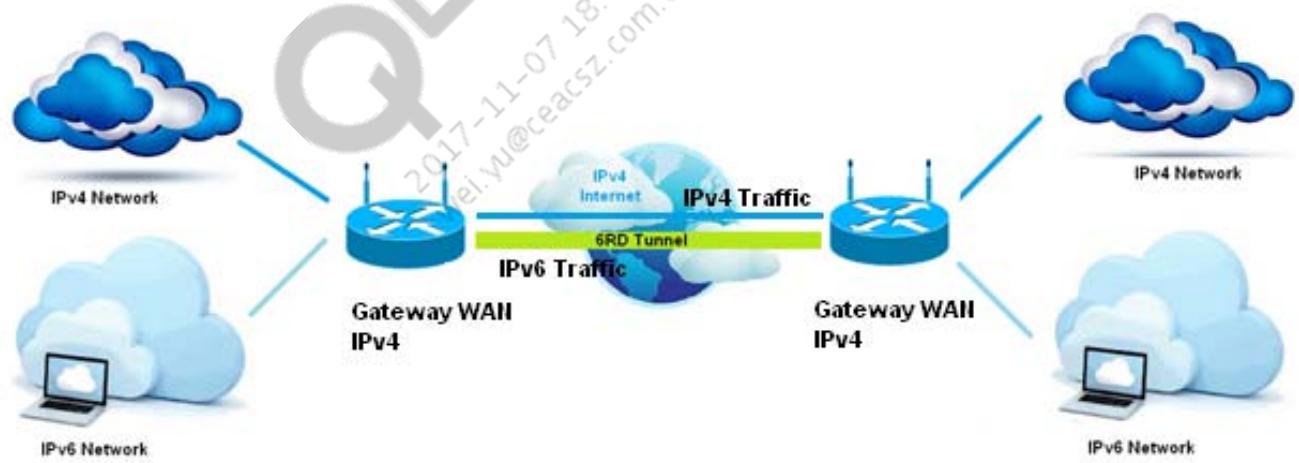
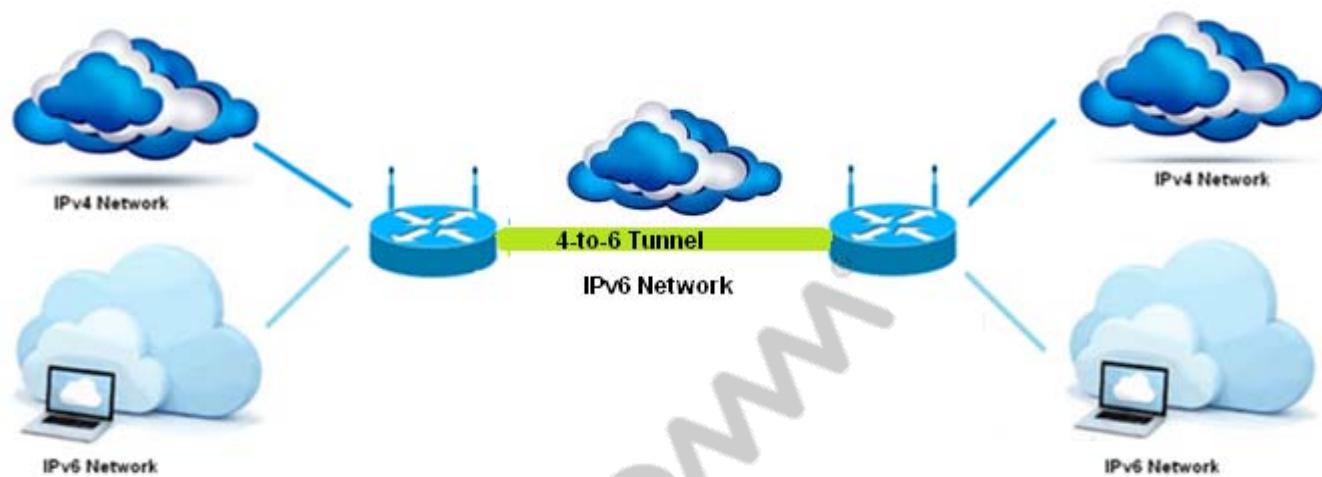


Figure 6-19 IPv6 Rapid Deployment (6RD) Mechanism



**Figure 6-20 Dual Stack Lite (DS-Lite) Mechanism - IPv4 Over IPv6**

### 6.19.2 Verify NSS acceleration

Use the following commands to verify the Fast Path for IPv4 TCP/UDP flow:

1. cat /sys/kernel/debug/qca-nss-drv/stats/ipv4
2. cat /sys/kernel/debug/qca-nss-connmgr-ipv4/connection\_stats

Use the following commands to verify the Fast Path for IPv6 TCP/UDP flow

1. cat /sys/kernel/debug/qca-nss-drv/stats/ipv6
2. cat /sys/kernel/debug/qca-nss-connmgr-ipv6/connection\_stats

### 6.19.3 Configuration commands

Check if following file exists: */proc/net/if\_inet6* to ensure that Kernel was built with IPv6 support. Also, there may be a need to load ipv6 module:

```
modprobe ipv6
```

On demand load is also possible by making following change to */etc/modprobe.conf*:

```
alias net-pf-10 ipv6
```

IPv6 like other Linux sub systems support multiple configuration options, extensions, experimental features etc. Proper configuration options need to be selected to build the kernel with required IPv6 features and extensions.

The following commands are used to configure interface, route and for diagnostic purpose and collecting dumps.

- ifconfig
- route

- ip
- ping6
- traceroute6
- tracepath6
- tcpdump
- radvd

### **6RD 6to4 Tunneling**

- ip tunnel add <tunnelname> mode sit local <localipv4address> ttl 64
- ip tunnel 6rd dev <tunnelname> 6rd-prefix <6rdprefix>/<6rdprefixlength>
- ip -6 addr add <ipv6addressoflocal tunnelendpoint>/<prefixlength> dev <tunnelname>
- ip link set <tunnelname> up
- ip -6 route add ::/0 via ::<borderrouterIPv4address> dev <tunnelname>

### **DS-Lite 4to6 Tunneling**

- ip -6 tunnel add <tunnelname> mode ipip6 remote <remoteipv6address> local <localipv6address> hoplimit 64
- ip link set dev <tunnelname> up
- ip route add default dev <tunnelname> metric 1

### **Other Commands**

- ip link set dev <interface> up
- ip link set dev <interface> down
- ip -6 addr show dev <interface>
- ip -6 addr add <ipv6address>/<prefixlength> dev <interface>
- ip -6 addr del <ipv6address>/<prefixlength> dev <interface>
- ip -6 neigh show [dev <device>]
- ip -6 neigh add <IPv6 address> lladdr <link-layer address> dev <device>

## **6.19.4 MAP-T Configuration**

### **DHCPv6**

#### **DUT:**

- Boot with factory default config
- Edit /etc/config/dhcp6c file and set as "option 'enabled' '1'"

**DHCPv6 server:**

- Install latest DHCPv6 server (sudo apt-get install isc-dhcp-server)
- Update /etc/dhcp/dhcpd.conf as follows

```
root@checstlc0015975-lin:~# cat /etc/dhcp/dhcpd.conf
ddns-update-style none;
default-lease-time 600;
max-lease-time 7200;
log-facility local7;
authoritative;
option dhcp6.map-option code 95 = string;
subnet6 7778::/64 {
    range6 7778::ff 7778::ffff;
    option dhcp6.name-servers 7778::1;
    option dhcp6.map-option
        00:59:00:16:00:0c:18:4e:4e:4e:00:30:77:78:00:00:00:00:00:5d:00:04:04:
        00:00:00:00:5b:00:09:40:77:78:00:00:00:00:ff:ff;
    prefix6 7778:: 7778:0:0:110:: /60;
}
```

- Start DHCPv6 by running the command 'dhcpd -6'

**Static****DUT:**

- Edit /etc/config/network as follows (Keep other configurations such as switch, switch\_vlan etc as is)

```
config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
    option ipaddr '127.0.0.1'
    option netmask '255.0.0.0'

config globals 'globals'
    option ula_prefix 'auto'

config interface 'lan'
    option ifname 'eth1'
    option type 'bridge'
    option proto 'static'
    option ipaddr '192.168.1.1'
    option netmask '255.255.255.0'
    option ip6assign '60'

config interface 'wan'
    option ifname 'eth0'
    option proto 'static'
```

```

option ip6addr '4aaa:db8::1/64'
option ip6gw '4aaa:db8::6'
option ip6prefix '4aaa:db8:0:1e0::/60'

config interface 'wan6_map'
    option proto 'map'
    option tunlink 'wan'
    option ip6prefix '4aaa:db8::'
    option ip6prefixlen '48'
    option ipaddr '172.16.16.0'
    option ip4prefixlen '24'
    option peeraddr '4aaa:db8:0:ff0::/64'
    option type 'map-t'
    option ealen '12'
    option psidlen '4'
    option offset '4'

```

- Restart network (/etc/init.d/network restart)
- Now when you send LAN to WAN IPv4 traffic on LAN, then you receive IPv6 on WAN
- For WAN to LAN, send the reverse traffic matching the upstream traffic (that is, Swap the source & destination IPv6 addresses, src/dst UDP ports and so on)

#### **BR:**

- Edit /etc/config/network as follows and restart network
 

```

root@OpenWrt:/# cat /etc/config/network

config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
    option ipaddr '127.0.0.1'
    option netmask '255.0.0.0'

config globals 'globals'
    option ula_prefix 'auto'

config interface 'lan'
    option ifname 'eth1'
    option type 'bridge'
    option proto 'static'
    option ip6addr 4aaa:db8::6/64'
    option ip6gw '4aaa:db8::1'

config interface 'wan6'
    option ifname 'eth0'
    option proto 'static'
    option ipaddr '192.168.137.10'
    option netmask '255.255.255.0'
    option ipgw '192.168.137.1'

config switch
    option name 'switch0'
    option reset '1'
    option enable_vlan '1'

```

```

config switch_vlan
    option device 'switch0'
    option vlan '1'
    option ports '6 1 2 3 4'

config switch_vlan
    option device 'switch0'
    option vlan '2'
    option ports '0 5'

```

- The preceding config assumes that eth1 is connected to DUT and eth0 to IXIA/Internet/Host
- Create and run the following script

```

root@OpenWrt:/# cat /etc/config/networkMAPT.sh
echo add mapNat46 > /proc/net/nat46/control
echo config mapNat46 remote.style MAP > /proc/net/nat46/control
echo config mapNat46 remote.v4 172.16.16.0/24 > /proc/net/nat46/control
echo config mapNat46 remote.v6 4aaa:db8::/48 > /proc/net/nat46/control
echo config mapNat46 remote.ea-len 12 > /proc/net/nat46/control
echo config mapNat46 remote.psid-offset 4 > /proc/net/nat46/control

echo config mapNat46 local.style RFC6052 > /proc/net/nat46/control
echo config mapNat46 local.v4 0.0.0.0/0 > /proc/net/nat46/control
echo config mapNat46 local.v6 4aaa:db8:0:ff0::/64 >
/proc/net/nat46/control
echo config mapNat46 local.ea-len 0 > /proc/net/nat46/control
echo config mapNat46 local.psid-offset 0 > /proc/net/nat46/control

echo config mapNat46 debug 0 > /proc/net/nat46/control

ifconfig mapNat46 up
ip route add 172.16.16.0/24 dev mapNat46
ip -6 address add 4aaa:db8:0:ff0::1/64 dev mapNat46

ip -6 route add 4aaa:db8::/48 via 4aaa:db8::1

ip route add default via 192.168.137.1

/etc/init.d/firewall stop

iptables -F
ip6tables -F
iptables -P FORWARD ACCEPT
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT

ip6tables -P FORWARD ACCEPT
ip6tables -P INPUT ACCEPT
ip6tables -P OUTPUT ACCEPT

/etc/init.d/firewall stop

```

### **Other configurations:**

- When LAN and WAN are swapped on DUT

```

config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
    option ipaddr '127.0.0.1'
    option netmask '255.0.0.0'

config globals 'globals'
    option ula_prefix 'auto'

config interface 'lan'
    option ifname 'eth0'
    option type 'bridge'
    option proto 'static'
    option ipaddr '192.168.1.1'
    option netmask '255.255.255.0'
    option ip6assign '60'

config interface 'wan'
    option ifname 'eth1'
    option proto 'static'
    option ip6addr '4aaa:db8::1/64'
    option ip6gw '4aaa:db8::6'
    option ip6prefix '4aaa:db8:0:1e0::/60'

config interface 'wan6_map'
    option proto 'map'
    option tunlink 'wan'
    option ip6prefix '4aaa:db8::'
    option ip6prefixlen '48'
    option ipaddr '172.16.16.0'
    option ip4prefixlen '24'
    option peeraddr '4aaa:db8:0:ff0::/64'
    option type 'map-t'
    option ealen '12'
    option psidlen '4'
    option offset '4'

```

■ Without NAT44 enabled

```

config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
    option ipaddr '127.0.0.1'
    option netmask '255.0.0.0'

config globals 'globals'
    option ula_prefix 'auto'

config interface 'lan'
    option ifname 'eth1'
    option type 'bridge'
    option proto 'static'
    option ipaddr '172.16.16.1'
    option netmask '255.255.255.0'
    option ip6assign '60'

```

```
config interface 'wan'
    option ifname 'eth0'
    option proto 'static'
    option ip6addr '4aaa:db8::1/64'
    option ip6gw '4aaa:db8::6'
    option ip6prefix '4aaa:db8:0:1e0::/60'

config interface 'wan6_map'
    option proto 'map'
    option tunlink 'wan'
    option ip6prefix '4aaa:db8::'
    option ip6prefixlen '48'
    option ipaddr '172.16.16.0'
    option ip4prefixlen '24'
    option peeraddr '4aaa:db8:0:ff0::/64'
    option type 'map-t'
    option ealen '8'
    option psidlen '0'
    option offset '0'
```

#### ■ PPPoE enabled on WAN

```
config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
    option ipaddr '127.0.0.1'
    option netmask '255.0.0.0'

config globals 'globals'
    option ula_prefix 'auto'

config interface 'lan'
    option ifname 'eth1'
    option type 'bridge'
    option proto 'static'
    option ipaddr '192.168.1.1'
    option netmask '255.255.255.0'
    option ip6assign '60'

config interface 'wan'
    option ifname 'eth0'
    option proto 'pppoe'
    option ipv6 1
    option username pppoe
    option password pppoe

config interface 'wan6'
    option ifname @wan
    option ip6addr '4aaa:db8::1/64'
    option ip6gw '4aaa:db8::6'
    option ip6prefix '4aaa:db8:0:1e0::/60'

config interface 'wan6_map'
    option proto 'map'
    option tunlink 'wan6'
```

```

option ip6prefix '4aaa:db8::'
option ip6prefixlen '48'
option ipaddr '172.16.16.0'
option ip4prefixlen '24'
option peeraddr '4aaa:db8:0:ff0::/64'
option type 'map-t'
option ealen '12'
option psidlen '4'
option offset '4'

```

## 6.20 IPv6 processing

IPv6 is an addressing mechanism suggested by IETF to take care of dwindling number of public IPv4 addresses available for users across the world. IPv6 address is 16 bytes (as compared to 4 bytes used for IPv4 address) and is expected to take care of future unique Layer 3 address requirements as we move towards an all IP digital world.

The following is a list of certain RFCs for IPv6:

RFC	Description
2460	Internet Protocol, Version 6 (IPv6) Specification
4861	Neighbor Discovery for IPv6
4862	IPv6 Stateless Address Auto configuration
3315	Dynamic Host Configuration Protocol for IPv6 (DHCPv6)
3633	IPv6 Prefix option for DHCPv6
3736	Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6
3646	DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)
3596	DNS Extensions to Support IP Version 6
3041	Privacy Extensions for Stateless Address Auto configuration in IPv6
4291	IPv6 Addressing Architecture
4193	Unique Local IPv6 Unicast Addresses
4293	Management Information Base for Internet Protocol (IP)
4443	Internet Control Message Protocol (ICMPv6) for the IPv6
2464	Transmission of IPv6 Packets over Ethernet Networks
1981	Path MTU Discovery for IPv6
3810	Multicast Listener Discovery Version 2 (MLDv2) for IPv6
4213	Basic Transition Mechanisms for IPv6 Hosts and Routers

2473	Generic Packet Tunneling in IPv6 Specification
2474	Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers
4038	Application aspects of IPv6 transition
2472/5072	IPv6 over PPP support
3596	Naming service support
5969	IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) -- Protocol Specification
6333	Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion
6674	Gateway-Initiated Dual-Stack Lite

The following are the salient features:

- Network discovery
  - Stateless auto configuration
  - IPv6 network prefix delegation
  - Router/Neighbor advertisements, solicitations and other optional messages and configurations
  - DHCPv6 server stateless and stateful discovery and configuration in LAN. DHCPv6 client on the WAN to receive IPv6 network prefix and router configurations
- Naming Service support (DNS servers)
 

Translation of IPv4 A record to IPv6 AAAA record and vice versa may be required. Queries need to be answered based on both IPv4 and IPv6 transport mechanisms.
- Multicast Listener Discovery (MLDv1/v2) snooping/proxy
 

Certain extensions have been added to MLD protocol to support IPv6.
- Firewall rules
 

Firewall rules based on fields from IPv6 header need to be supported.
- Fragmentation
 

Fragmentation by intermediate routers is not supported in IPv6.

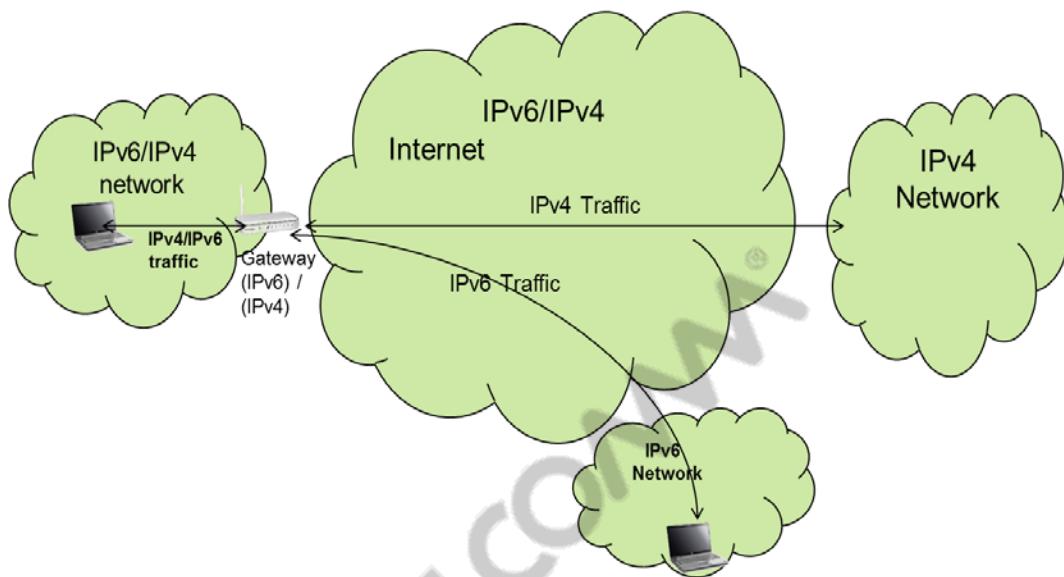
### 6.20.1 IPv4 to IPv6 transition on WAN link

#### Native Dual Stack

Native Dual Stack is the ideal solution for IPv4 to IPv6 transition provided enough IPv4 addresses are around.

Two separate IPv4 and IPv6 stacks or some kind of hybrid stack (single interface with both IPv4 and IPv6 addresses) are used to provide both IPv4 and IPv6 support. The DHCP and DNS services must support both IPv6 and IPv4 network types.

The following figure demonstrates native dual stack based deployment.



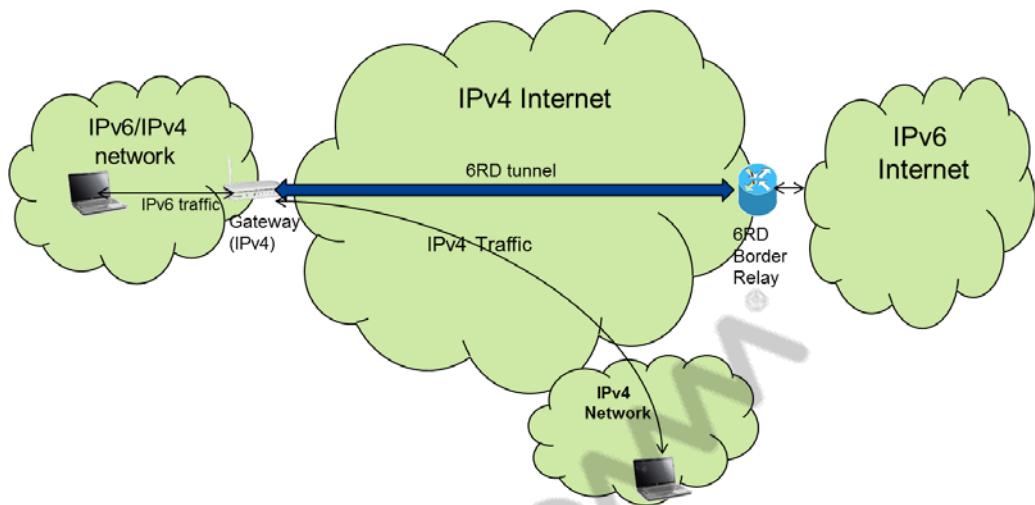
**Figure 6-21 Native dual stack co-existence mechanism**

## 6RD

6RD provides IPv6 network services over an IPv4 network using a tunnel to the 6RD border relay gateways.

6RD border gateways use IPv4 anycast addresses for failover/resiliency. DHCP 6RD extensions provide options to support tunneling end point IPv4 address and the gateway network prefix configuration. Host IPv6 addresses are formed in such a way that the mapping between the IPv6 host address and the IPv4 tunnel can be done without holding any state of the connection. MTU of the tunnel should be configured properly so that fragmentation can be avoided. ICMPv4 errors need to be mapped to ICMPv6 errors and forward to the IPv6 host in the LAN.

The following figure demonstrates 6RD-based deployment.



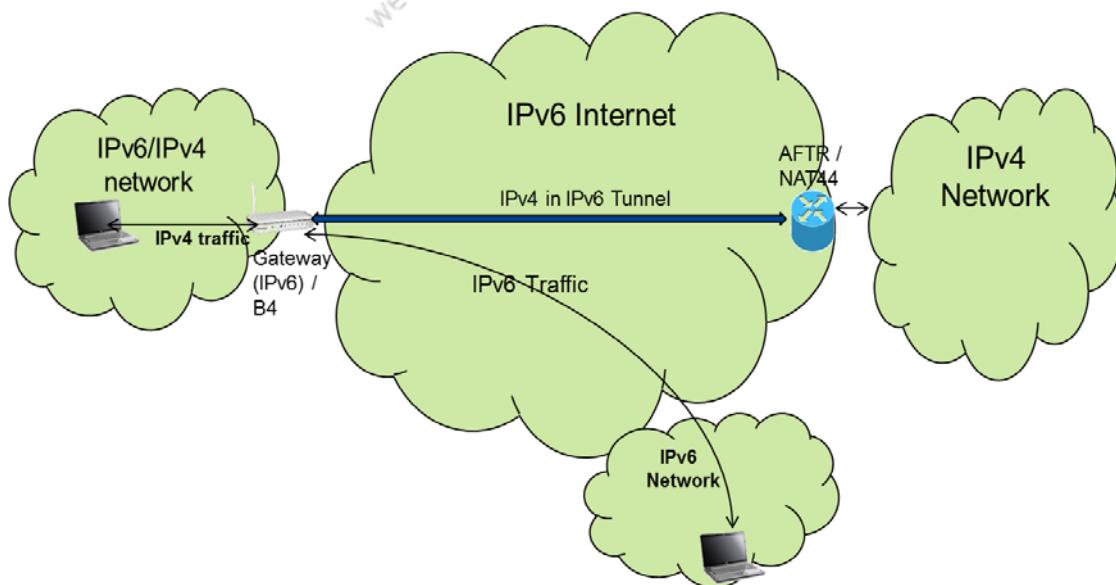
**Figure 6-22 6RD coexistence mechanism**

### Dual Stack Lite

Dual Stack Lite is used to provide IPv4 services to host behind the IPv6 ISP network and IPv6 addressable gateway. It uses IPv4 in IPv6 tunnel.

MTU of the tunnel should be configured properly. Otherwise it will lead to fragmentation. ISP issues private IPv4 addresses to subscribers through the tunnel and uses Large Scale NAT (LSN) to communicate with the IPv4 Internet.

The following figure demonstrates DS-Lite based deployment:



**Figure 6-23 Dual stack lite coexistence mechanism**

## Configuration commands

Check if following file exists: `/proc/net/if_inet6` to ensure that Kernel was built with IPv6 support.  
Also, there may be a need to load ipv6 module:

```
modprobe ipv6
```

On demand load is also possible by making following change to `/etc/modprobe.conf`:

```
alias net-pf-10 ipv6
```

IPv6 like other Linux sub systems support multiple configuration options, extensions, experimental features etc. Proper configuration options need to be selected to build the kernel with required IPv6 features and extensions.

Following commands are used to configure interface, route and for diagnostic purpose and collecting dumps.

`ifconfig`, `route`, `ip`, `ping6`, `traceroute6`, `tracepath6`, `tcpdump`, `radvd`

### 6RD 6to4 tunneling

- `ip tunnel add <tunnelname> mode sit local <localipv4address> ttl 64`
- `ip tunnel 6rd dev <tunnelname> 6rd-prefix <6rdprefix>/<6rdprefixlength>`
- `ip -6 addr add <ipv6addressoflocal tunnelendpoint>/<prefixlength> dev <tunnelname>`
- `ip link set <tunnelname> up`
- `ip -6 route add ::/0 via ::<borderrouterIPv4address> dev <tunnelname>`

### DS-Lite 4to6 tunneling

- `ip -6 tunnel add <tunnelname> mode ipip6 remote <remoteipv6address> local <localipv6address> hoplimit 64`
- `ip link set dev <tunnelname> up`
- `ip route add default dev <tunnelname> metric 1`

### Other commands

- `ip link set dev <interface> up`
- `ip link set dev <interface> down`
- `ip -6 addr show dev <interface>`
- `ip -6 addr add <ipv6address>/<prefixlength> dev <interface>`
- `ip -6 addr del <ipv6address>/<prefixlength> dev <interface>`
- `ip -6 neigh show [dev <device>]`
- `ip -6 neigh add <IPv6 address> lladdr <link-layer address> dev <device>`

## 6.21 VLAN traffic separation over Wi-Fi

This section describes the VLAN traffic Separation over Wi-Fi. By default, no VLAN Tag is seen in the air when we configure VLAN interfaces on AP and STA VAP. The objective is to get 12 byte VLAN Tag (along with LLC Support).

Packets in the air will have 12 byte VLAN + LLC Header containing information as follows:

Frame Format: {dst\_addr[6], src\_addr[6], 802.1Q header[4], EtherType[2], Payload...}

LLC + VLAN header is just before the Ethernet Type field.

LLC: {DSAP [1], SSAP [1], Control Field [1], Organization Code [3], 802.1Q Virtual LAN [2]}

VLAN: {VLAN ID [2], EtherType [2]}

### 6.21.1 Sample configuration scenarios

This section describes configuration on AP and STA vap to see tagged Packet in the air. Host Driver controls VLAN tagged packet sent out by Network Stack using Dev->flags. No separate host-FW communication is required for the same. These flags can be controlled using iwpriv athx wlan\_tag 0/1. 0 will switch off the flag, thus Wi-Fi device is advertised as having VLAN\_HW\_TX, VLAN\_HW\_RX capability. Hence, VLAN tag is not seen over the air in this case. On the other hand, iwpriv athx wlan\_tag 1 erases these flags and so tagged packets will be seen in the air.

#### 6.21.1.1 Configure VLAN Tag for Single VAP on AP and STA

1. Bring up AP and STA on channel 36, mode 11ACVHT80.
2. Use ifconfig tool to verify settings on AP & STA.
3. Let STA associate to AP.
4. Run following commands on both AP and STA to get VLAN tag over Wi-Fi (12 Byte LLC mode support)

```
iwpriv ath<x> wlan_tag 1
```

```
vconfig add ath<x> y
```

```
ifconfig athx.y X.X.X.X up
```

5. Run traffic from AP to STA and sniff packets in the air using Wireshark.

12 Byte LLC + VLAN (8 + 4) will be seen in the air before IP information.

6. Non-Tagged can also be seen by disabling device features as follows:

```
vconfig rem ath<x>.y (Remove previous VLAN interface, if present)
```

```
iwpriv ath<x> wlan 0
```

```
vconfig add ath<x>.y
```

```
ifconfig athx.y X.X.X.X up
```

7. Run traffic from AP to STA and sniff packets in the air using Wireshark.

- No LLC or VLAN tag is seen.
8. Similarly, backend to backend traffic can be run by configuring VLAN on Ethernet of back ends of AP and STA.

### 6.21.1.2 Configure VLAN Tag for Multi-VAP

1. Bring up multiple VAPs on AP on channel 36, mode 11ACVHT80.
2. Connect multiple STAs to VAPs.
3. Configure `vlan_tag` to be 1 and 0 on alternate AP VAPs and corresponding STAs connected to it.

```
AP1-STA1: iwpriv ath<x> vlan_tag 1
           vconfig add ath<x> y
           ifconfig athx.y X.X.X.X up
```

```
AP1-STA1: iwpriv ath<x> vlan_tag 0
           vconfig add ath<x> y
           ifconfig athx.y X.X.X.X up
```

4. Run Iperf traffic between AP and STA.
5. Packets between AP1-STA1 should have 12 byte LLC+VLAN header.
6. Packets between AP2-STA2 should not have VLAN header.
7. Similarly, backend to backend traffic can be run by configuring VLAN on Ethernet of back ends of AP and STA.

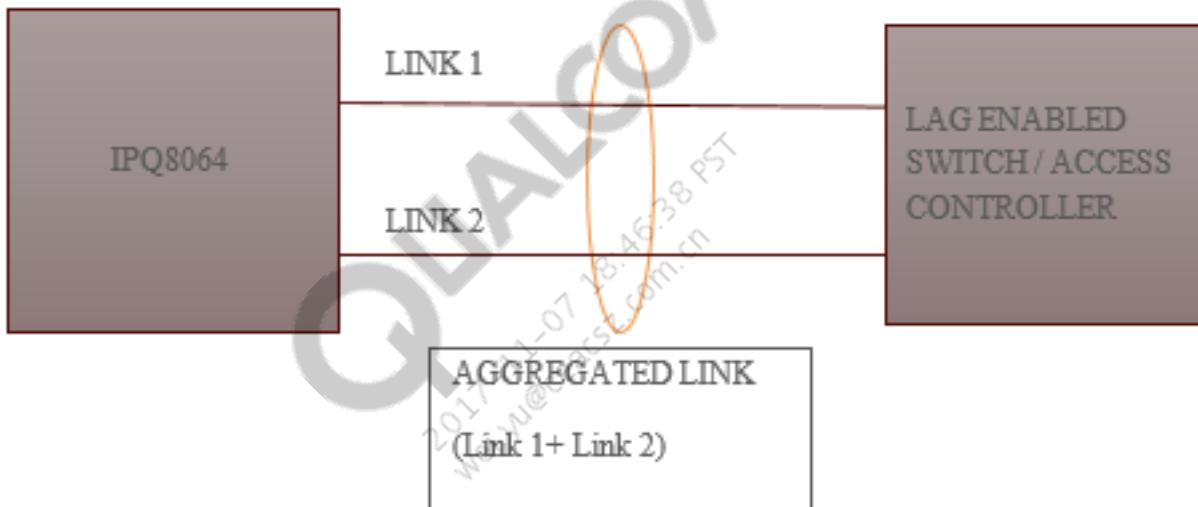
### 6.21.1.3 Configure VLAN Tag for Multicast Traffic

1. Bring up AP and STA1 and STA2 on channel 36, mode 11ACVHT80.
  2. On AP, issue following Command to set multicast mode to translate:
  3. `iwpriv athx mcastenhance 2`
  4. Let stations connect to AP.
  5. Set `vlan_tag` to 1 on AP, STA1 and STA2 and configure VLAN interfaces on ath.
- ```
iwpriv ath<x> vlan_tag 1
           vconfig add ath<x> y
           ifconfig athx.y X.X.X.X up
```
6. Run multicast traffic using iperf/chariot from AP to STA1 and STA2 (over VLAN interface).
  7. Look for LLC + VLAN Tag in the air.
  8. Similarly, test with setting `vlan_tag` to 0 on AP, STA1 and STA2 and running multicast traffic over VLAN interfaces. VLAN tag should not be in this configuration.

## 6.22 Link aggregation group

The IPQ8064 SoC is a next generation SoC for advanced network packet processing. It is a router/smart gateway processor chip and has a Dual-core Krait application processor, Network subsystem with two Ubi-32 cores, ARM9 RPM processor and various peripheral interfaces for providing connectivity. This section describes the support for Link Aggregation (LAG) on IPQ8064.

Link Aggregation is used to aggregate multiple physical links, connected to the same link partner, to increase throughput beyond what a single link can sustain. MAC clients can treat the aggregated group of links as if it were a single link. This allows improved utilization of available links in bridged LAN environments, along with improved resilience in the face of failure of individual links.



**Dynamic Link Aggregation:** Defined in IEEE 802.1ax or the previous IEEE 802.3ad. The selection of port to be used for transmitting a frame is done by frame distribution or load balancing algorithms. In doing so it may provide load balancing. Note that a flow can be shifted to another link for load balancing or due to exceptions. However, a flow cannot use more than one link simultaneously. The specification does not specify any load balancing algorithm but mandates that it must not allow

1. Misordering of frames
2. Duplication of frames

**Link Aggregation and Control Protocol (LACP):** LACP is part of IEEE802.3ad/1.ax specification and provides a method to control the bundling of several physical ports together to form a single logical channel. LACP allows a network device to negotiate an automatic bundling of links by sending LACP packets to the peer (directly connected device that also implements LACP).

*Static Link Aggregation:* Static Link Aggregation does not follow IEEE 802.1ax/.3ad and does not use LACP.

*Linux Bonding Driver:* The Linux bonding driver provides a method for aggregating multiple network interfaces into a single logical "bonded" interface to be used by MAC clients. Physical interfaces are configured to become slaves of this bonding master. For each frame to be egressed from a bonded interface, the driver uses load balancing algorithms to select a physical slave interface for the frame to be egressed on.

Linux bonding driver supports both dynamic and static modes for an aggregation. It supports various Tx hash policies for these modes such as 'Layer2' (uses L2 addresses), 'Layer2+3' (uses L2+L3 addresses), 'Layer3+4' (uses L3+L4 addresses) algorithms. Out of these, the first two are IEEE 802.1ax compliant. It also stateless and does not keep track of flows and takes a LAG decision for each packet to be egressed. On link failure, flows using that link are shifted to other links. On a link up, flows that were originally on this link are shifted back to this link again in addition to other flows that may select this link.

On IPQ8064, link aggregation will be used to bundle Ethernet links to an access controller carrying 11ac and 11n wireless LAN traffic and achieve 1.5 Gbps bandwidth assuming 4x4 11ac with 80 MHz channel bandwidth.

### 6.22.1 LAG requirements

- Maximum of four aggregation/bond groups.
- Support static link aggregation (balance-xor mode of Linux bonding driver)
- Support dynamic link aggregation (802.3ad mode of Linux bonding driver) and Link Aggregation and Control Protocol (LACP) for dynamic link aggregation.
- No restriction on the number of GMACs that can be part of a bundle. However, ports connected directly to S17 switch may not be used in an aggregation.
- Load balance different flows on different links assuming Tx hash policy selects different links for different flows.
- Dynamically reallocate a flow to a different port in case of link failure.
- Frame ordering per flow must be maintained.
- No duplication of transmitted frames.
- Utilize all links in an active aggregation, assuming Tx hash policy distributes different flows to different links.
- For outgoing frames, support the following frame distribution / load balancing algorithms:
  - Layer 2: XOR of hardware MAC addresses to generate hash
  - Layer 2+3: Uses combination of Layer 2 and Layer 3 addresses to generate hash
  - Layer 3+4: Uses combination of Layer 3 and Layer 4 addresses to generate hash. Note that this is not compliant to IEEE 802.3ad/.1ax.

Linux bonding driver supports all of the aforementioned frame distribution algorithms.

No support to transition flows to different links using Marker Protocol for dynamic link aggregation.

Features supported are limited to those supported by Linux bonding driver unless there is a good reason to extend the same (for example, changes required to support fast path).

## 6.22.2 LAG architecture

A LAG decision is taken for an outgoing packet when load balancing algorithms are executed. On IPQ8064, the load-balancing algorithm executes on HLOS and hence aggregation decision is taken by HLOS. These algorithms use source and destination addresses in packet headers to determine a physical interface to be used to egress this packet. This result is used by ECM to push appropriate rules to NSS. The ECM itself only sees a LAG interface as exported by the link aggregation driver of HLOS, but uses the link aggregation driver to translate an output LAG interface to an output physical interface for packets being egressed from the LAG interface. Hence, rules pushed to NSS use physical interface. Virtual interface identifiers are not used.

Once a rule is pushed, subsequent packets are routed/bridged in NSS fast path effectively performing link aggregation in fast path. From the point of view of NSS, there is no difference between rules pushed with or without LAG.

## 6.22.3 HLOS updates

The following components in HLOS are updated to support LAG on IPQ8064:

1. Enhanced Connection Manager: NSS rules are pushed and destroyed from the ECM. The ECM needs to identify the egress interface as a LAG master and determine the slave physical interface to be used. It also manages ports being set as active or inactive by the bonding driver due to operation of LACP protocol or link failure.
2. Linux Bonding (LAG) Driver: APIs will be added to this driver that will apply the currently configured load balancing algorithm to a packet and return the physical slave interface to be used to egress the same.

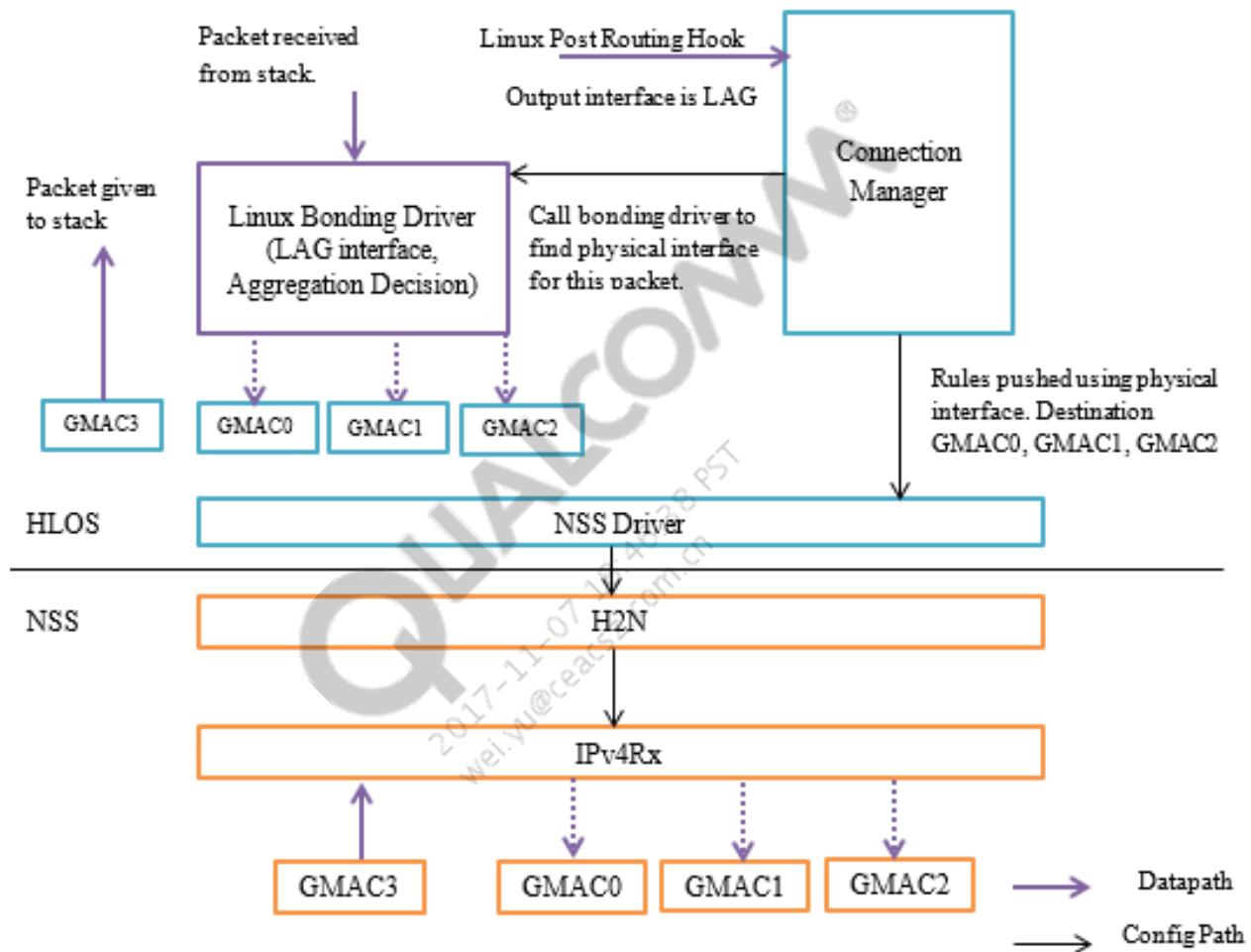
## 6.22.4 Rule push to NSS

The following steps describe the sequence involved when a new rule is pushed to NSS:

1. When a packet is received by NSS through a GMAC and there is no rule that exists for that flow in the NSS rule table, that packet is sent via exception path to host.
2. A forwarding decision is taken by host, typically based on L3/L4 addresses. This decision may select a LAG interface as the egress interface for this packet.
3. On receiving this packet through the Linux Post Routing Hook, ECM checks to see if the egress interface is a LAG [master] interface. If so, it calls into Linux bonding driver, which uses load balancing algorithms, to determine the physical (LAG slave) interface that this packet would be egressed on. Note that this interface is one of the slaves of the master interface selected by the HLOS forwarding path. ECM then uses this interface to push an appropriate rule to NSS.

**NOTE** Although forwarding decision is based on L3/L4 addresses, the LAG decision will select the interface and the slave of the interface is selected by forwarding path.

The following figure illustrates a typical flow of how a LAG based rule is pushed to NSS.



## 6.22.5 Destruction of rules

Rules are destroyed by ECM when it receives a destroy event in its connection tracking notifier callback. This process is same as when any other non-LAG related rule is destroyed. The lookup to find these rules does not use egress interface as a parameter. Hence the type of interface does not affect rule deletion.

## 6.22.6 Link-down event

The process of a LAG slave interface link failure will be handled in the following manner:

1. When a LAG slave interface link goes down the port state is subsequently set as inactive by bonding driver.
2. Bonding driver will communicate this to ECM that will destroy corresponding LAG based rules using that interface

## 6.22.7 Link-up event

The process of a LAG slave interface link up will be handled in the following manner:

1. After a LAG slave interface link comes up the port state is subsequently set as active by bonding driver
2. Bonding driver will communicate this to ECM
3. ECM will delete corresponding rules using that interface causing flows to flow through host.
4. New rules will be pushed for all rules that start flowing through host.

This allows CM to rebalance flows using the recent upped link.

## 6.22.8 NSS Sync Notifications

NSS driver receives sync and stats notifications from the NSS. It passes them onto CM using a callback. These callbacks aim to keep conntrack connections alive. Updating of conntrack statistics does not involve or is influenced by interface type. Hence, no LAG specific updates to CM's statistics sync callback is required. Moreover, these statistics do not need to be communicated to bonding driver as it is stateless and does not maintain statistics for flows.

## 6.22.9 LAG interface up/down

When a LAG interface is administratively set up/down, it will cause the same state change to all slave interfaces. When a LAG interface is put down, connection entries for flows using the slave interfaces will time out. They will then be deleted by CM's conntrack callback. This mechanism enables a customer to support proprietary load balancing algorithm. Updates are required in Linux Bonding Driver.

## 6.22.10 Design

This subsection provides details of design and implementation of link aggregation as applicable for support on IPQ8064. The following modules need to be added or updated for the same.

Support in Enhanced Connection Manager for LAG can be aptly divided into support for routing and bridging.

**Interface Type Combinations:** The ECM must deal with combinations of virtual interfaces comprising bridge, vlan and LAG, and find the NSS specific physical interface number.

Updates are made to ECM post routing hook. Before pushing a L3 rule, the connection manager finds the NSS specific interface number for the egress interface.

This interface is checked for being a LAG interface and the physical interface determined for the same.

```
If ((dest_dev->priv_flags & IFF_BONDING) && (dest->priv_flags & IFF_MASTER)) {  
    /* Call bonding driver to get physical interface */  
    /* Call NSS driver to get interface number for physical interface*/  
}
```

The ECM can use the retrieved interface number to push necessary rules. If the egress interface is not a LAG, the ECM continues to process the packet as required.

The following APIs are added to Linux Bonding Driver to support link aggregation.

```
Struct net_device *bond_get_tx_dev(struct sk_buff *skb, void *src_mac, void *dst_mac,  
void *src, void *dst, uint32_t protocol ,  
struct net_device *bond_dev, __be16 *l4hdr)
```

This function applies the load balancing algorithm, currently configured for this bond interface, on the packet to be egressed and returns the physical interface that will be used for the same. It returns pointer to interface of the LAG slave or NULL if no suitable interface can be found. The API arguments are described as follows:

Skb: sk\_buff pointer

Src\_mac: L2 source address

Dst\_mac: L2 destination address

Src: L3 source address

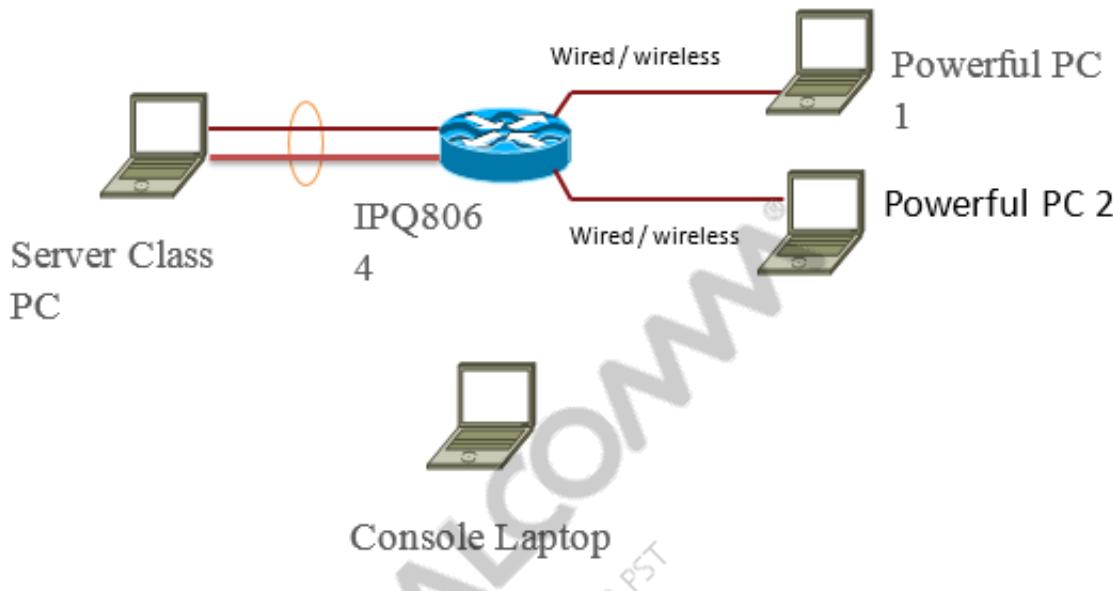
Dst: L3 destination address

Protocol: L3 protocol type

Bond\_dev: Pointer to LAG interface structure

l4hdr: Pointer to Layer 4 header

### 6.22.11 Feature Test Setup – TCP/UDP



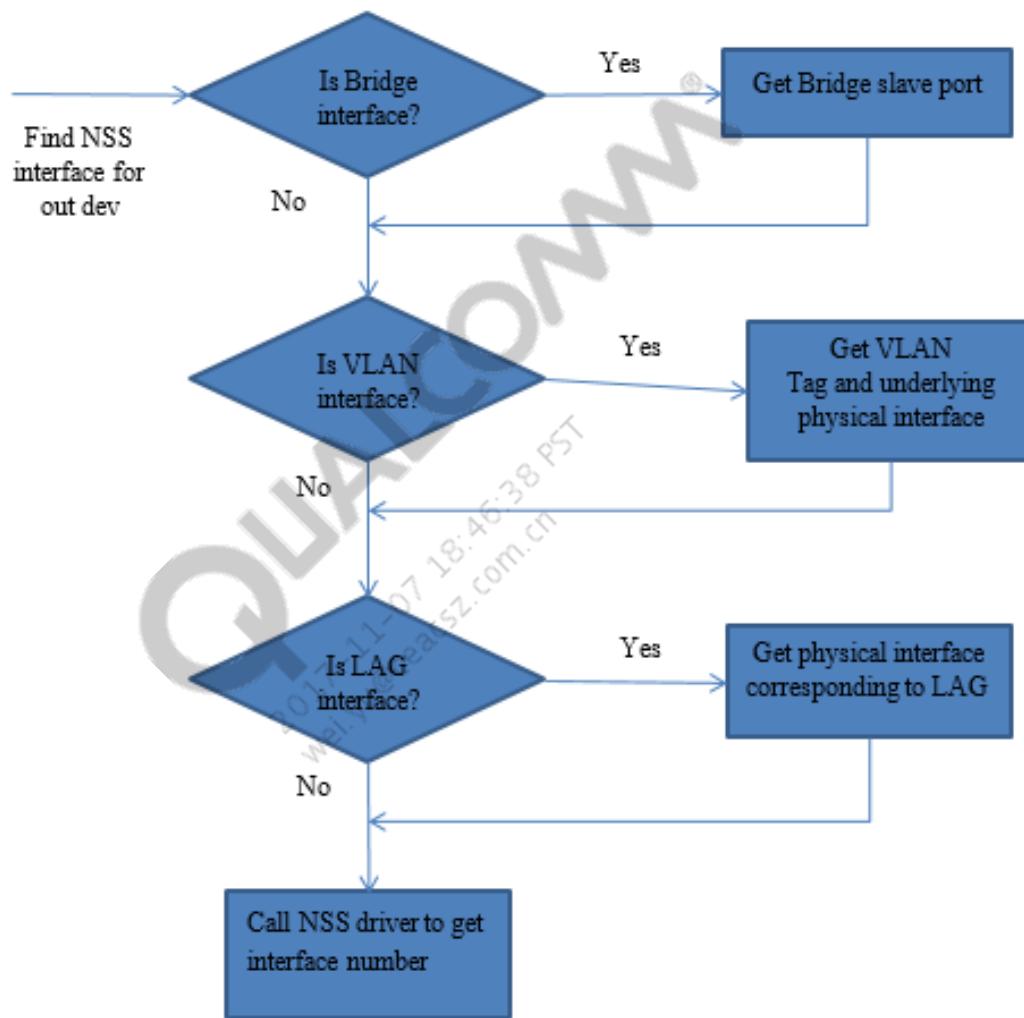
### 6.22.12 Linux Bonding Driver Options

The following options are provided by the Linux Bonding Driver to configure link aggregation.

1. mode: Specifies one of the bonding policies.
  - a. 802.3ad or 4
  - b. balance-xor or 2
2. lacp\_rate: Option specifying the rate in which we'll ask our link partner to transmit LACPDU packets in 802.3ad mode. Possible values are:
  - a. slow or 0: Request partner to transmit LACPDUs every 30 seconds
  - b. fast or 1: Request partner to transmit LACPDUs every 1 second
3. xmit\_hash\_policy: Selects the transmit hash policy to use for slave selection in balance-xor and 802.3ad modes. Possible values are:
  - a. layer2: Uses XOR of hardware MAC addresses to generate the hash.
  - b. Layer2+3: This policy uses a combination of layer2 and layer3 protocol information to generate the hash.
  - c. Layer3+4: This policy uses a combination of layer3 and layer4 protocol information to generate the hash.
4. miimon: Specifies the MII link monitoring frequency in milliseconds.

### 6.22.13 Determine the type of egress interface

1. The connection manager needs to be aware that an egress interface may be any one of the following virtual interfaces: Bridge, VLAN or LAG. It thus needs to check the type of egress virtual interface and find the corresponding physical interface. The intended order of this check is depicted in the following diagram:



## 6.23 LACP framework

**NOTE** LACP is supported on IPQ8064 chipsets only.

Link Aggregation Control Protocol (LACP) as defined in IEEE 802.3ad, provides additional functionality for link aggregation groups (LAGs). Use the link aggregation feature to aggregate one or more Ethernet interfaces to form a logical point-to-point link, known as a LAG, virtual link, or bundle. The MAC client can treat this virtual link like a single link.

Link aggregation increases bandwidth, provides graceful degradation as failure occurs, and increases availability. It provides network redundancy by load-balancing traffic across all available links. If one of the links should fail, the system automatically load-balances traffic across all remaining links.

When LACP is not enabled, a local LAG might attempt to transmit packets to a remote single interface, which causes the communication to fail. When LACP is enabled, a local LAG cannot transmit packets unless a LAG with LACP is also configured on the remote end of the link.

A typical LAG deployment includes aggregate trunk links between an access switch and a distribution switch or customer edge (CE) device.

Daemon LACPD run in user space, and is responsible for LACP protocol handling. LACPD run one LACP protocol instance for every switch. LACPD daemon read configuration from file “/etc/config/lacpd” to create ports and aggregators. Every switch in the configuration file has its own LACP protocol instance. LACP protocol will associate each port to an aggregator which may include only one port. LACPD creates a trunk in switch for every aggregator whose members are more than one. So switch port whose associated aggregator has only one member will act as normal port, and can be connected to PC.

LACPD daemon open a raw socket to receive LACP packet from Linux kernel. This raw socket listen on ether type 0xfefe which represent Atheros header. LACPD daemon will find out the port according to the receiving Linux network device and the port information in the Atheros header. Receiving Linux network device can be got by the source address of the ‘recvfrom’ function.

Same raw socket is used to transmit LACP packet on switch port. An Atheros header which include the real output switch port should be inserted into LACP packet before transmitting. This LACP packet should be transmitted on Linux interface which connect to the external hardware switch. Please note that this interface may be also in a Linux bonding. You should directly use this interface rather than its bonding interface to send LACP packet. You can refer to this kind of Linux network device as a control channel of switch ports.

Trunk and trunk members are configured with switch driver.

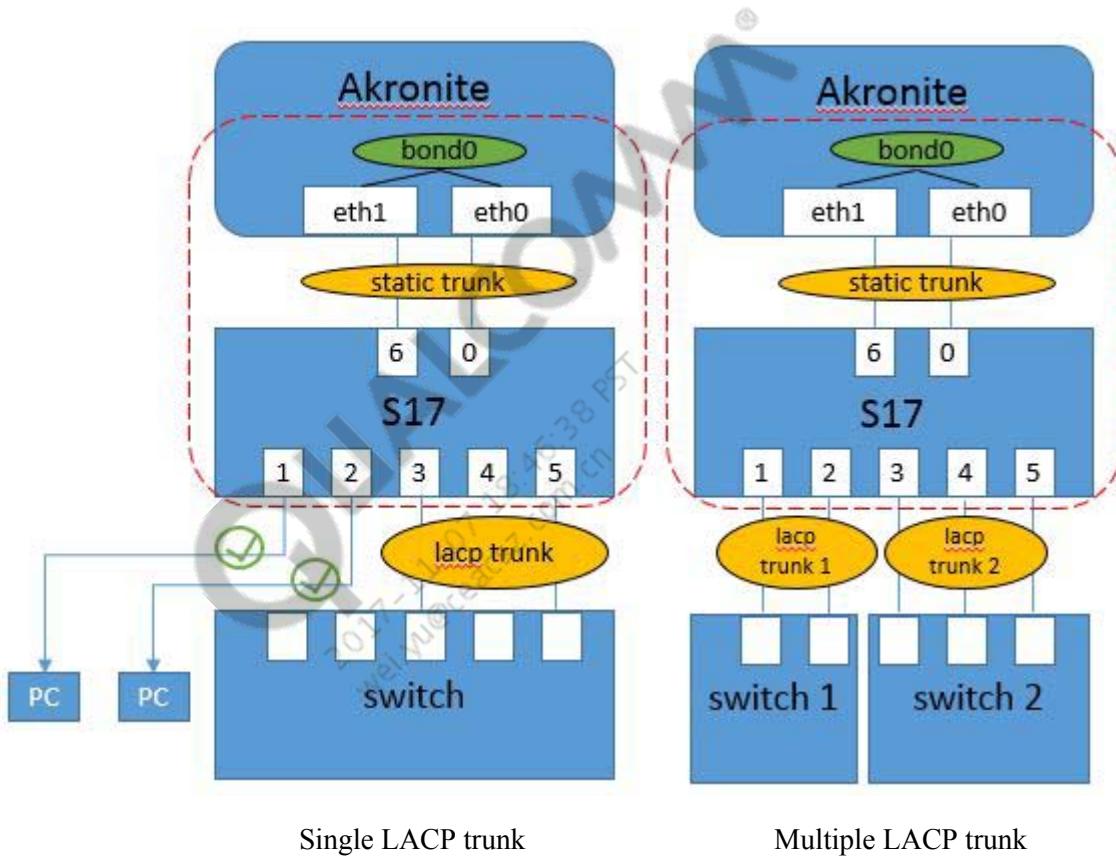
To prevent possible loop before trunk creation, we will disable switch port until it is added into an aggregator. This may bring 2 seconds delay before switch port is really able to use. LACPD daemon need to monitor link status of every switch port. Once a switch port becomes down, LACPD should reconstruct corresponding trunk.

Some other applications like STP/IGMP may also want to know switch port information. Trunk in switch will impact them, so LACPD should provide some service API to get trunk status:

| API  | Description   |
|--|---|
| <code>ubus call lacpd get_trunk &lt;switch&gt; &lt;port&gt;</code> | Convert port id to trunk id. Return port id itself if port is not a trunk member.   |
| <code>ubus call lacpd get_port &lt;switch&gt; &lt;trunk&gt;</code> | Convert trunk id to port id. Normally it will return port id of first trunk member. |

|                               |  |
|-------------------------------|--|
| /etc/hotplug.d/switch/qc<br>a | Scripts to be executed when old port is not available.<br>SWITCH="s17", PORT="id", ACTION="portdown" |
| /etc/hotplug.d/switch/qc<br>a | Scripts to be executed when new port is available.<br>SWITCH="s17", PORT="id", ACTION="portup"       |

The following is a sample configuration scenario for LACP:



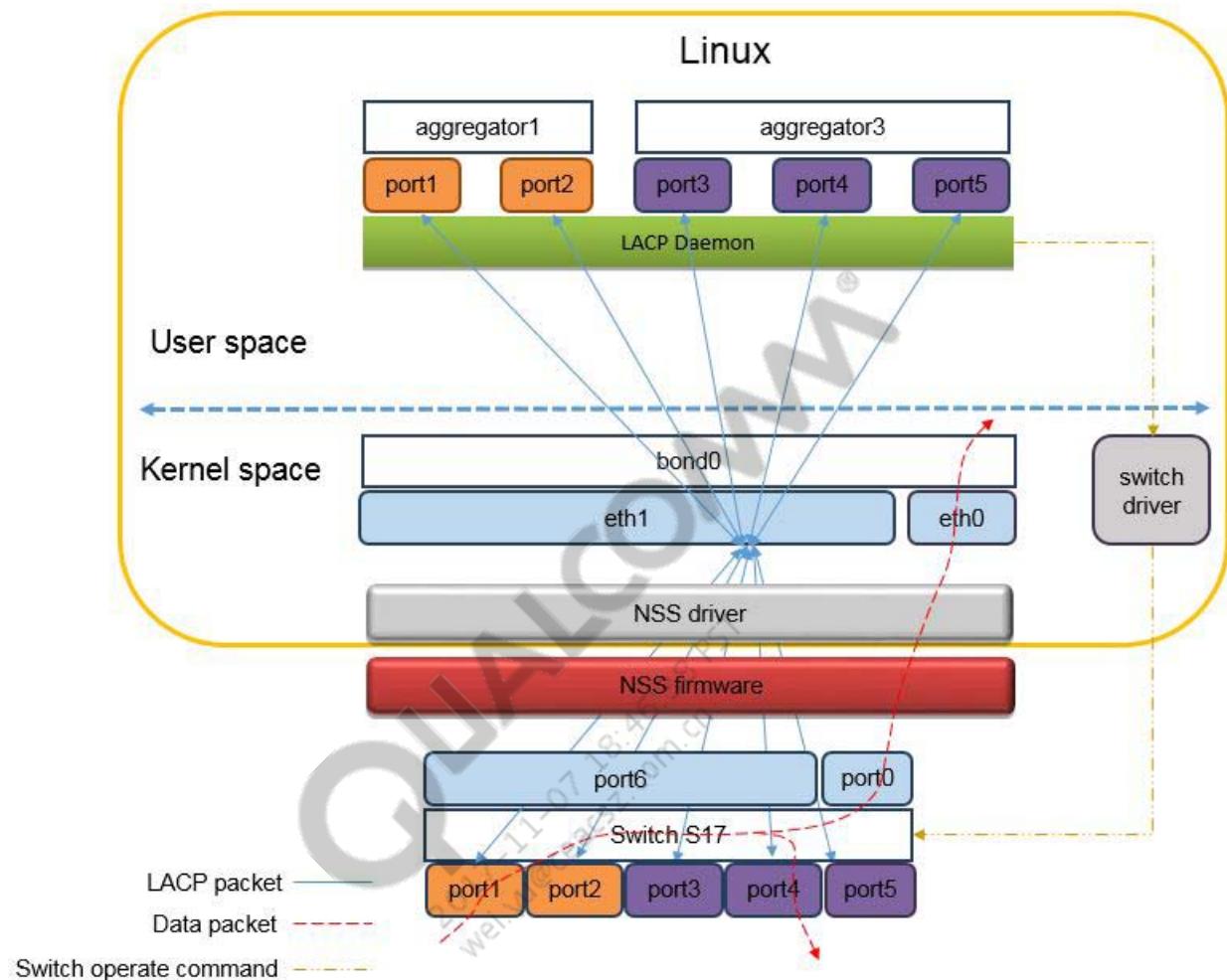
In this use case GMAC eth0 is connected to port 0 of switch S17, GMAC eth1 is connected to port 6 of switch S17, ports 0, 1, 2, 3, 4, 5, 6 are in the same VLAN, for example VLAN 1.

Firstly we need to create a static trunk on ports 0 and 6 in switch, as a mapping, we also need to create a static bonding on Linux interfaces eth0 and eth1.

Secondly we can select some ports to create LACP trunks. For example, we can use ports 1, 2 to create LACP trunk 1, and use ports 3, 4, 5 to create LACP trunk 2. Other unused ports can be still connected to PC.

Please note that all ports 0,1,2,3,4,5,6 should be in the same VLAN in this use case.

The following framework illustrates LACP:



## 6.23.1 User interface of link aggregation

### 6.23.1.1 Start/stop link aggregation

To start link aggregation, enter the `/etc/init.d/lacpd start` command.

To stop link aggregation, enter the `/etc/init.d/lacpd stop` command.

### 6.23.1.2 UCI configuration interface

Switch link aggregation can be configured by UCI configuration file “`/etc/config/lacpd`”

| option           | Value range | Description |
|------------------|-------------|-------------|
| Section : switch |             |             |

|                |  |  |
|----------------|--|--|
| ifname         | Name of Linux network device             | Linux network device which represent the switch  |
| Section : lacp |  |  |
| enable         | 0   1                                    | Enable lacp instance, user can define multiple instance  |
| name           | string                                   | Name of lacp instance  |
| txHashPolicy   | “L2”   “L3”   “L2_L3”                    | Hash policy to select output port when transmit packet   |
| lacpRate       | 0   1                                    | Lacp packet sending rate,<br>0 : slow, 1 per 30 seconds; 1 : fast, 1 per 1 second.   |
| adSelect       | “all”   “stable”   “bandwidth”   “count” | All: all aggregator are active<br>Stable: select only one aggregator, keep current active aggregator if it is still useable<br>Bandwidth: select only one aggregator which has biggest bandwidth<br>Count: select only one aggregator which has the most members |
| Section : port |  |  |
| enable         | 0   1                                    | Enable or disable lacp on this port  |
| lacp           | Name of lacp section                     | Specify a lacp instance which this port belongs to   |
| switch         | Name of switch section                   | Specify a switch which this port belongs to  |
| switchPortId   | 1-4                                      | Mapped port number in the switch   |
| userKey        | 0-65535                                  | User defined operation key, default is 0   |

The following is an example for file “/etc/config/lacpd”

```

config switch S17
    option ifname 'eth1'

config lacp lan
    option enable '1'
    option name 'lan'
```

```
option txHashPolicy 'L2_L3'  
option lacpRate '0'  
option adSelect 'all'  
  
config port port1  
    option enable '1'  
    option lacp 'lan'  
    option switch 'S17'  
    option switchPortId '1'  
    option userKey '0'  
  
config port port2  
    option enable '1'  
    option lacp 'lan'  
    option switch 'S17'  
    option switchPortId '2'  
    option userKey '0'  
  
config port port3  
    option enable '1'  
    option lacp 'lan'  
    option switch 'S17'  
    option switchPortId '3'  
    option userKey '0'  
  
config port port4  
    option enable '1'  
    option lacp 'lan'  
    option switch 'S17'  
    option switchPortId '4'  
    option userKey '0'
```

### 6.23.2 Status of link aggregation

Use the following command to obtain the LACP status:

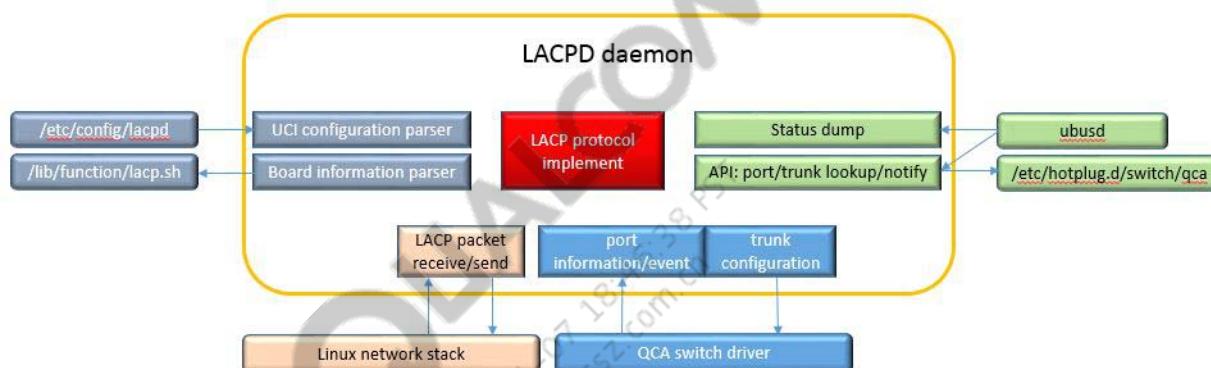
```
ubus call lacpd status <name of lacp instance>
```

### 6.23.3 User space link aggregation daemon

#### 6.23.3.1 Code base

User space daemon LACPD use Linux bonding driver as code base. Linux bonding driver is a full implementation of LACP protocol in Linux source code “drivers/net/bonding/bond\_3ad.c”. This LACP protocol implementation is ported into LACPD daemon as LACP core. Both Linux bonding driver and LACPD daemon is under GPLv2 license.

#### 6.23.3.2 Components



#### 6.23.3.3 UCI script

UCI script “`/etc/init.d/lacpd`” is the user interface. User use command “`/etc/init.d/lacpd start`” to start LACP, and use command “`/etc/init.d/lacpd stop`” to stop LACP. LACPD will read configuration from file “`/etc/config/lacpd`” when it starts. LACPD also execute script “`/lib/function/lacp.sh`” in its initial stage to get some board specific information.

#### ubusd

LACPD exports two types of API to ubusd. One is to dump LACP status. User can use command “`ubus call lacpd status`” to dump status of LACP. Another is to do translation between port and trunk. Other applications such as STP/IGMP can use command “`ubus call lacpd get_trunk <switch> <port>`” or command “`ubus call lacpd get_port <switch> <trunk>`” to do translation between port and trunk.

#### Hot plug script

When a port is added to or removed from a trunk, LACPD will execute all scripts under directory “`/etc/hotplug.d/switch/qca`” to notify other applications which need this event. Following environment variables will be passed to scripts depending on event type.

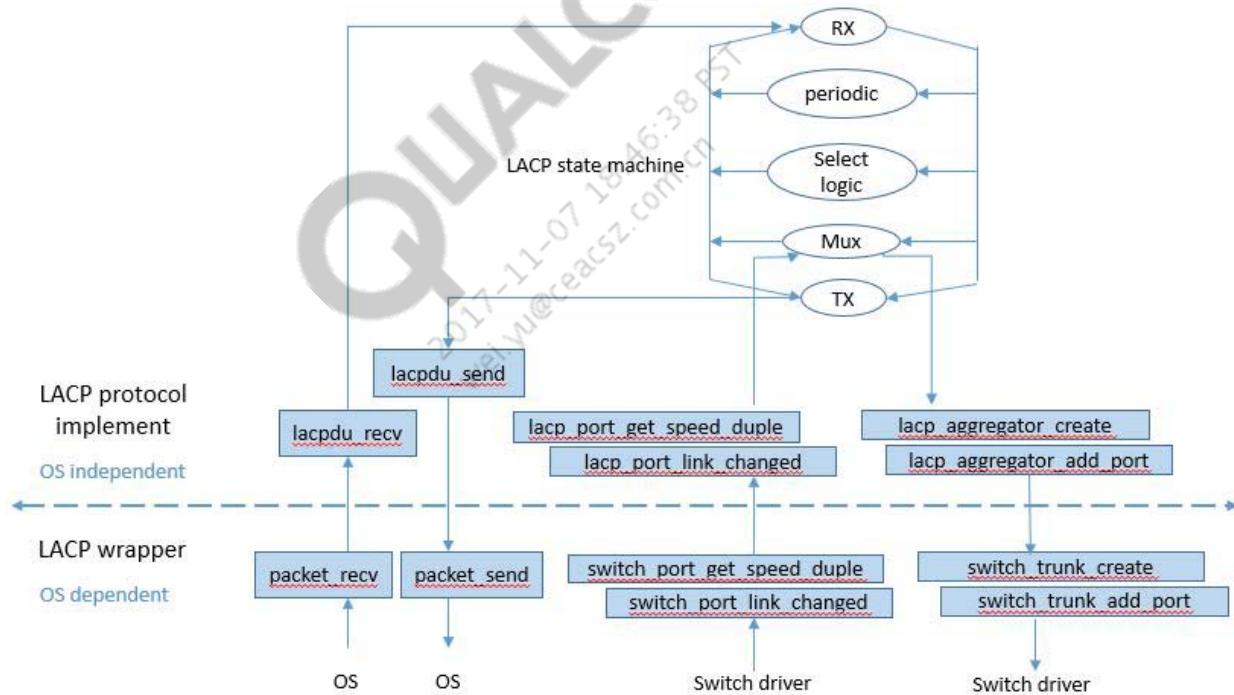
| Event | Variables |
|-------|-----------|
|-------|-----------|

|                       |  |
|-----------------------|--|
| Port is available     | SWITCH="s17", PORT="id", ACTION="portup"<br>When a port is available, id is in range "0-7". When a trunk is available, id is in range "256-263".           |
| Port is not available | SWITCH="s17", PORT="id", ACTION="portdown"<br>When a port is not available, id is in range "0-7". When a trunk is not available, id is in range "256-263". |

### 6.23.3.4 LACPD daemon

LACPD daemon “/sbin/lacpd” implement all LACP protocol algorithm.

### 6.23.4 Framework of LACPD daemon



#### ■ Trunk id

A new trunk in switch is considered as a logical link. You should treat trunk as the same with normal port, especially in applications which be aware of switch port, such as STP/IGMP. So we have to assign an id to trunk, then other applications can operate on it. Because normal port id range is 0-7, so we define trunk id range as 256-65535. You can simply right rotate 8 bits on trunk id to get a real trunk id in switch.

#### ■ Lookup API

| Parameter | Description |
|-----------|-------------|
|-----------|-------------|

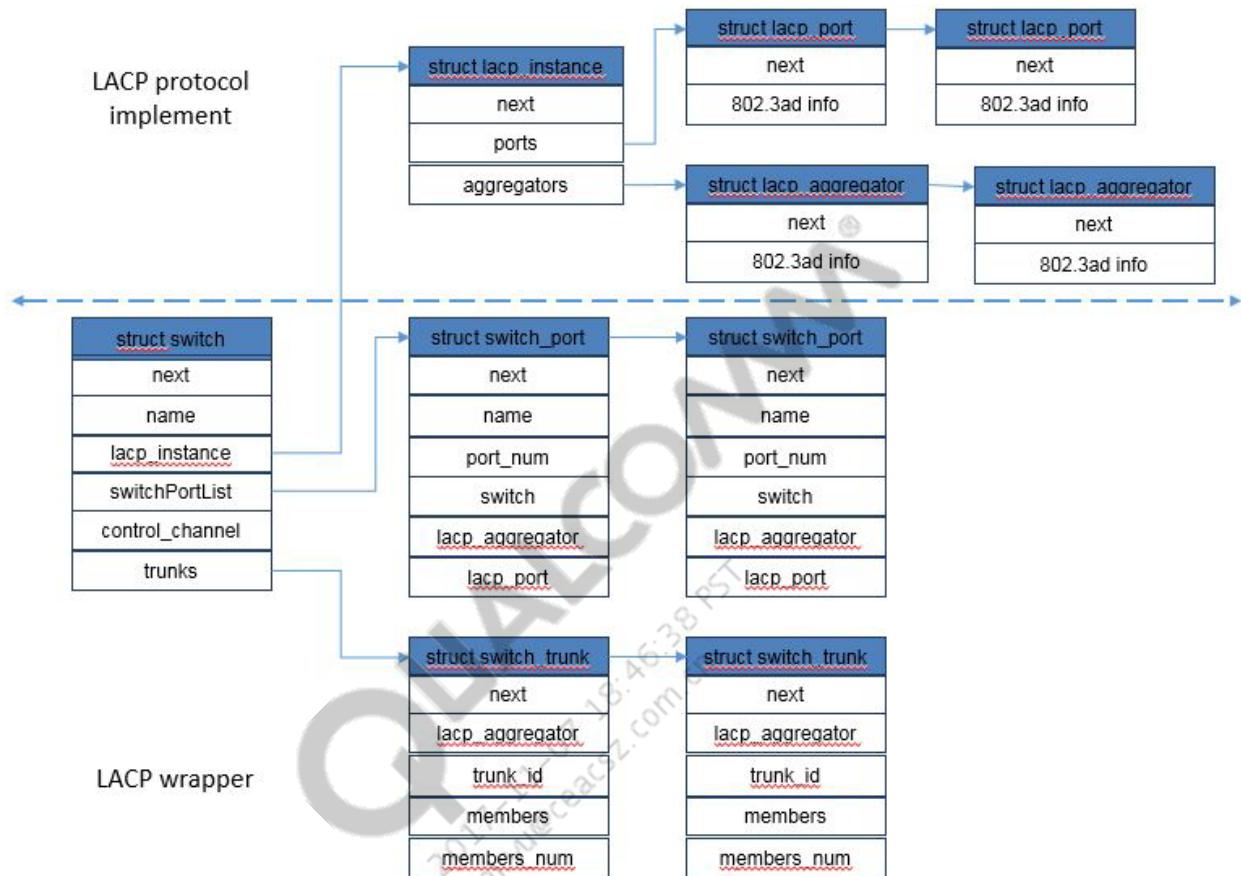
---

```
struct switch * switch_find_by_name(char *name)
name           Name of switch, defined in UCI configuration
struct switch * switch_find_by_interface(char *if_name)
If_name        Name of linux interface, Linux connect to external hardware switch by
               this interface.
struct switch_port *switch_port_find(struct switch *sw, int port_num)
sw            Pointer of a structure of switch
Port_num      Id of port
struct switch_trunk * switch_trunk_find_by_port(struct switch *sw, int port_num)
sw            Pointer of a structure of switch
port          id of port
struct switch_trunk * switch_trunk_find_by_aggregator(struct lacp_aggregator *ag)
ag           Find out which switch trunk is associated with this aggregator
```

---

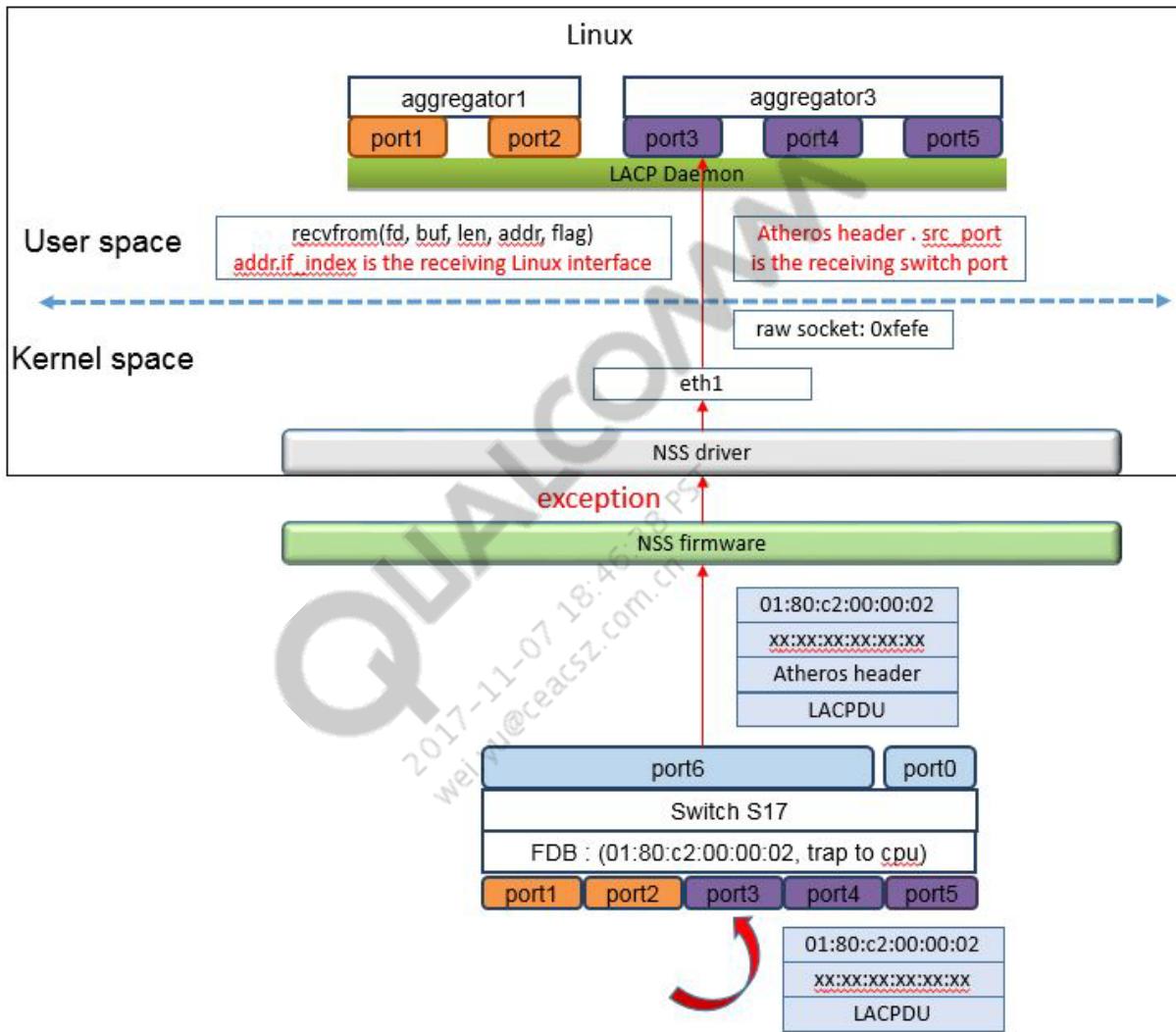
QUALCOMM  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

### 6.23.5 Data structure of LACPD daemon



## 6.23.6 Receive and transmit LACP packet

### Receive LACP packet



LACPD daemon open a raw socket to listen on 0xfefe packet type. This packet type identify a LACP packet encrypted with an Atheros header. The fifth parameter ‘src\_addr.sll\_ifindex’ of function ‘recvfrom’ contains receiving Linux interface when LACP packet reached LACPD daemon. This Linux interface connect to an external hardware switch. It represent a switch section in configuration file “/etc/config/lacpd”. Received LACP packet contains an Atheros header, and the original receiving switch port is identified by the source port in the Atheros header. LACPD daemon find out the switch port according to switch name and switch port num.

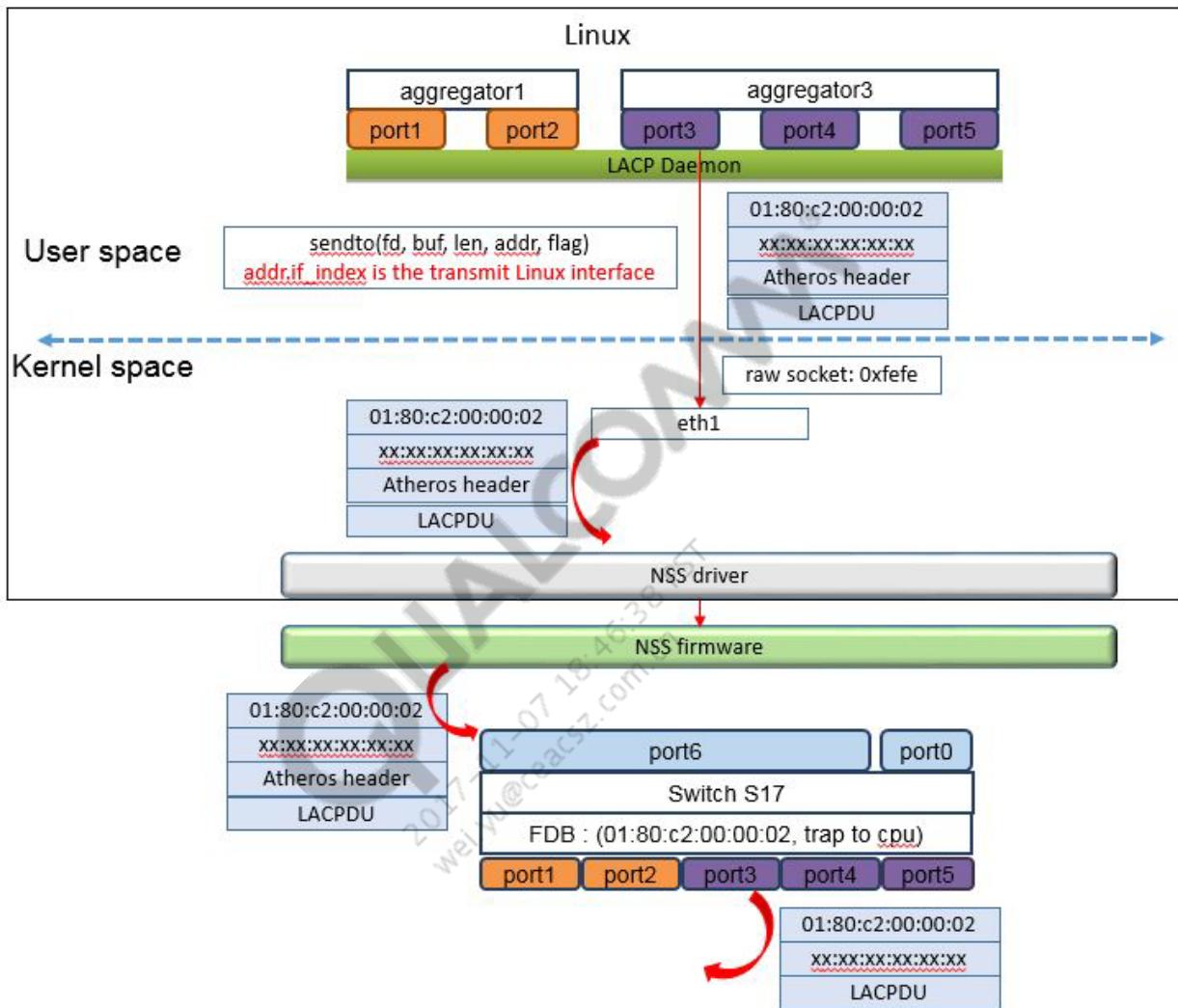
Switch should trap all LACP packets to its CPU port with Atheros header added.

The following API can be used to add a reserved FDB in switch to trap LACP packet to Linux.

|                    |  |
|--------------------|--|
| <b>Command</b>     | <b>ssdk_sh port hdrtype set enable 0xfefe</b>  |
| <b>Description</b> | Set ether type of Atheros header to 0xfefe   |
| <b>Command</b>     | <b>ssdk_sh port rxhdr set 0 onlymanagement</b>   |
| <b>Description</b> | Allow receiving management packet with Atheros header on port 0  |
| <b>Command</b>     | <b>ssdk_sh port txhdr set 0 onlymanagement</b>   |
| <b>Description</b> | Inserting Atheros header when sending management packet on port 0  |
| <b>Command</b>     | <b>ssdk_sh port rxhdr set 6 onlymanagement</b>   |
| <b>Description</b> | Allow receiving management packet with Atheros header on port 6  |
| <b>Command</b>     | <b>ssdk_sh port txhdr set 6 onlymanagement</b>   |
| <b>Description</b> | Inserting Atheros header when sending management packet on port 6  |
| <b>Command</b>     | <b>ssdk_sh mirror analyPt set 6</b>  |
| <b>Description</b> | Mirror BPDU to port 6 which is connected to eth1   |
| <b>Command</b>     | <b>ssdk_sh fdb resventry add 01-80-c2-00-00-02 65535 rdtcpu forward</b>  |
|                    | null y n y n n y n   |
| <b>Description</b> | Use a reserved FDB entry to redirect and mirror LACP packet to port 0<br>and port 6 regardless the stp state of receiving port |

QUALCOMM  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

### 6.23.6.1 Transmit LACP packet



LACPD daemon uses the same raw socket which is used to receive LACP packet from switch port to transmit LACP packet on switch port. Transmitted LACP packet is a raw packet, it contains all necessary headers, such as MAC header. Transmitted LACP packet also include an Atheros header, the real transmitting switch port number is included in this Atheros header. When sending LACP packet on switch port, LACPD daemon will find out associated switch and get the Linux interface which connect to the hardware switch this switch port reside in. The real transmitting Linux interface is identified by the fifth parameter ‘`dst_addr.sll_ifindex`’ of function ‘`sendto`’. Switch will strip Atheros header and send this LACP packet to port specified in Atheros header.

When a packet with Atheros header was received on switch CPU port, switch should send it to switch port which is specified in Atheros header.

## 6.23.7 Trunk configuration

### 6.23.7.1 Switch driver API to add/delete trunk/port

Switch driver should provide following API to manipulate trunk in hardware switch:

| Command Description | <b>ssdk_sh trunk group set &lt;trunk_id&gt; &lt;enable/disable&gt; &lt;port bitmap&gt;</b>   |
|---------------------|--|
|                     | Argument <i>trunk id</i> specify which trunk to be created, its range is 1-4.<br><i>port bitmap</i> specify the trunk members. <i>Enable</i> means create a trunk, <i>disable</i> means destroy a trunk. |

### 6.23.7.2 Switch driver API to set transmit hash mode

| Command Description | <b>ssdk_sh trunk hashmode set &lt;hash_mode&gt;</b>  |
|---------------------|--|
|                     | Argument <i>hash mode</i> is a 4 bit number. Bit 0 means destination mac address, bit 1 means source mac address, bit 2 means destination IP address, bit 3 means source IP address. Once a bit is set in the hash mode, corresponding packet field will participate hash calculation, |

## 6.23.8 Requirement for hardware switch about port state

1. Receive LACP packet when switch port is disabled

To prevent possible loop before trunk creation, we will disable switch port until it is added into an aggregator. But we still need to receive LACP packet even when switch port is disabled. Switch should identify LACP packet and trap it to Linux.

2. Do not forward LACP packet to other ports except CPU port

LACP packet should be trapped to Linux, and should not be forwarded to other ports

## 6.23.9 Link status notification

### 6.23.9.1 Notify link status of switch port

Because switch driver didn't provide NETLINK message for link status of switch port, so LACPD daemon have to poll the link status of switch port at a configurable interval which is 3 seconds by default.

### 6.23.10 Failover in link aggregation

Because LACPD has to poll link status of switch port, so there must be some delay to detect link failure of switch port. So switch hardware should provide a fail-over mechanism to detect link status of switch port, and move traffic from failed port to ports which is still up.

### 6.23.11 Integration with other subsystem

#### 6.23.11.1 VLAN support

VLAN can still be added on a switch port even port is in a link aggregator. But we restrict that all members in an aggregator must be in the same VLAN. Otherwise behavior will be unpredictable.

#### 6.23.11.2 Spanning tree

Spanning tree should operate on trunk rather than trunk member. Once a port is added into a trunk, STP should remove this port from its database. Conversely, STP should add a port into its database once this port is removed from a trunk. Similarly, STP should treat trunk as a normal port, so add it to its database once a trunk is created, and remove it from its database once a trunk is destroyed.

#### 6.23.11.3 IGMP

Like spanning tree, IGMP also need to treat trunk as a normal port, and update its database when trunk is created or destroyed.

### 6.23.12 Limitations and Notice

#### 6.23.12.1 Limitation: Source mac address of LACP packet cracked

Because of S17 switch hardware limitation, LACP packet will be sent base on hash when its output port is a trunk member. So we have to crack source mac address of LACP packet to let it be sent on a specific switch port.

#### 6.23.12.2 Limitation: Link status notification delay

Because switch driver did not provide link status notification of switch port to user space, so LACPD daemon have to poll the link status of switch port every 3 seconds, this behavior will delay the responding when topology change. There is a parameter to adjust this interval, it is in line 7 of “/lib/functions/lacp.sh”, but it is not opened to user.

#### 6.23.12.3 Notice: Mirror function used

LACP used mirror function of switch. This function is a global configuration. Other module should not use it again, otherwise LACP will be impacted.

## 6.24 802.3x flow control

IEEE 802.3x is an Ethernet flow control mechanism that is implemented at the link layer. It allows the congested station to send PAUSE frames. When the overwhelmed network node sends a PAUSE frame, the sender needs to halt the transmission for a period of time, which is specified in the frame. The sender device can be a host station or another switch. Either way, the sender needs to either buffer the packets or drop them when it receives a PAUSE frame.

802.3x protocol is not priority-based, thus does not support CoS. PAUSE functionality is not per VLAN or per class. For layer 2 QoS to be honored, flow control needs to be turned off.

### 6.24.1 Requirements

The sole purpose of implementing 802.3x for GMACs is to prevent the packet drops during slow path traffic. Previously when slow path traffic was being sent at a rate higher than Kraits can handle, the excess packets were being dropped before handoff to Kraits. This feature enables enterprise customers using slow path not to sustain packet drops before Kraits.

PAUSE functionality is implemented for flows that are not accelerated by NSS, that is, flows where NSS hands the packets to Host. However, if there is a mix of accelerated and non-accelerated flows originating from the same port, both will get affected since the sender will pause transmission of all packets. If all the flows flowing through the port are of accelerated nature, they will not be subject to flow control.

### 6.24.2 Implementation

#### Rx direction

Receive flow control will be limited to detecting pause frames and stopping our transmission at GMAC. In other words, NSS RX flow control feature will be contained in Synopsys GMAC, and there will be no propagation of the pause frame to the user application (NSS firmware) or action taken by NSS.

Rx flow control can be enabled by ethtool. Please see the Application Notes for the ethtool commands.

Ethtool controls the following register values in GMAC:

**Table 6-7 Rx MAC Flow Control**

| RFE | DM | Description   |
|-----|----|---|
| 0   | x  | The MAC receiver does not detect the received Pause frames. |

|          |          |   |
|----------|----------|---|
| <b>1</b> | <b>0</b> | The MAC receiver does not detect the received Pause frames but recognizes such frames as Control frames.            |
| <b>1</b> | <b>1</b> | The MAC receiver detects or processes the Pause frames and responds to such frames by stopping the MAC transmitter. |

### RFE Receive Flow Control Enable:

When this bit is set, the MAC decodes the received Pause frame and disables its transmitter for a specified (Pause) time. When this bit is reset, the decode function of the Pause frame is disabled.

#### Tx direction

In enterprise solutions, the handoff from NSS firmware to HLOS constitutes the slowest part of the pipeline. To avoid dropping frames during this handoff, n2h will send a signal to all GMAC ports when its queue reaches the high watermark. Upon receiving the signal, GMACs will transmit a PAUSE frame by setting the pause frame bit in their corresponding TX flow control registers.

A pause frame from a GMAC can be generated in two ways:

1. TFE bit is set which indicates Transmit Flow control is enabled, and the RX FIFO threshold is reached. In this scenario, GMAC automatically transmits a pause frame.
2. You write 1 to FCB\_BPA bit of the flow control register. When operating in full duplex mode, this triggers GMAC to send a pause frame without regards to its rx FIFO queue.

As explained, the aforementioned approach 2 is used to generate pause frames by using FCB\_BPA bit.

#### Flow Control Busy or Backpressure Activate Bit (FCB\_BPA)

This bit initiates a Pause frame in the full-duplex mode and activates the backpressure function in the half-duplex mode if the TFE bit is set. In the full-duplex mode, this bit should be read as 1'b0 before writing to the Flow Control register. To initiate a Pause frame, the Application must set this bit to 1'b1. During a transfer of the Control Frame, this bit continues to be set to signify that a frame transmission is in progress. After the completion of Pause frame transmission, the MAC resets this bit to 1'b0. The Flow Control register should not be written to until this bit is cleared.

### 6.24.3 Configure 802.3x flow control

The flow control feature can be enabled by using ethtool. Once the flow control for an interface is enabled, in order for this to take effect, it needs to be followed by a restart of the said interface.

Example:

```
# enable flow control for eth2
ethtool -A eth2 tx on rx on
# restart eth2
ifconfig eth2 down; ifconfig eth2 up
```

```
# enable flow control for eth3
ethtool -A eth3 tx on rx on
# restart eth3
ifconfig eth3 down; ifconfig eth3 up
```

Once the flow control is enabled and the system sees traffic more than it can handle, GMACs will start sending pause frames to the responsible port. Eg. in the case of eth2 to eth3 high pps unidi traffic, eth2 will generate pause frames and send them to PC1.

PC1 – eth2 – dut – eth3 – PC2

One can then verify that pause frames by eth2 were generated by issuing the following command:

```
ethtool -S eth2 | grep tx_pause_frames
```

**NOTE** PC1 needs to be configured to honor these pause frames for flow control to actually take effect. For an IXIA port, this means checking flow control under port configuration.

In case of bidirectional traffic that exceeds Kraits processing limits, both ports would send pause frames even if one of the flows is sent at a lower rate. NSS does not have the capability to recognize the offending port, both eth2 and eth3 traffic is merged in one pool before handoff. This can again be verified by checking the tx\_pause\_frames statistic for both ports using ethtool.

```
ethtool -S eth2 | grep tx_pause_frames
ethtool -S eth3 | grep tx_pause_frames
```

Disable the flow control by following commands. Similar to enable, it needs to be followed by a restart of the interface.

```
# disable flow control for eth2
ethtool -A eth2 tx off rx off
ifconfig eth2 down; ifconfig eth2 up

# disable flow control for eth3
ethtool -A eth3 tx off rx off
ifconfig eth3 down; ifconfig eth3 up
```

## 6.25 LAN port ID handling NSS and IPQ40x8 platforms

On IPQ40x8 platforms, for SFE traffic acceleration and its connection track, SFE dump displays only 5 tuple information and does not contain switch port ID information. With the LAN port ID handling NSS and IPQ40x8 switch, SFE dump displays port ID of switch from which an

accelerated flow is received and also contains the port ID on which the flow is transmitted. Apart from the display of port ID, SFE dump is not modified. ECM dump will show hierarchy, which includes ethx id. Note that each GMAC has unique Ethernet netdevice interface name. Display of the port ID information in SFE dumps can be analyzed by verifying the routing and bridging between combination of ports (for example, eth0 to br-lan (eth1/eth2/eth3/eth4) routing and eth1 to eth2 bridging with different GMAC ports) and by validating the WLAN to eth0/br-lan connection.

The switch port name or number is displayed for each flow that is accelerated. The traffic acceleration features are applicable as follows.

- Supported for traffic acceleration on both IPQ8064 and IPQ401x platforms (VLAN, QoS, Multi Flow acceleration, NAT, IPv6)
- Supported for traffic acceleration only on IPQ401x platforms (SFE-HNAT Co Existence)
- Not supported on IPQ401x platforms (supported for traffic acceleration - static flow)

### 6.25.1 EDMA Multiple VLANs

This section describes sample scenarios for EDMA Driver with support for multiple VLANs, where-in the existing implementation limits supported number of VLAN groups to two. With this implementation, the available switch ports can be grouped into any number of possible VLAN groups, with the number limited only by the number of physical ports available.

The implementation takes the number of VLANs supported as well as the group configurations from dtsi file, and creates a lookup table which maps the port-id from the switch to the vlan-id of the groups created, and forwards packets to the relevant interfaces to linux.

### 6.25.2 Sample Topology

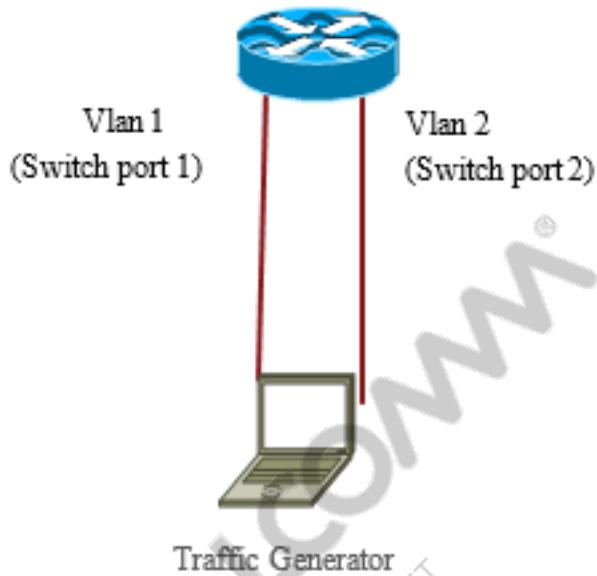


Figure 6-24 Sample Topology

### 6.25.3 2 VLANs (1+4) The default case

Configuration Steps

- i) DTSI config

File: qcom-ipq40xx.dtsi

```
aliases {  
    spi0 = &spi_0;  
    spi1 = &spi_1;  
    i2c0 = &i2c_0;  
    i2c1 = &i2c_1;  
    ethernet0 = "/soc/edma/gmac0";  
    ethernet1 = "/soc/edma/gmac1";  
};  
edma@c080000 {  
    . . .
```

```
qcom,rx_head_buf_size = <1540>;  
qcom,num_gmac = <2>;  
gmac0 {  
    local-mac-address = [000000000000];  
    qcom,poll_required = <0x1>;  
    qcom,phy_mdio_addr = <4>;  
    qcom,forced_speed = <1000>;  
    qcom,forced_duplex = <1>;  
    vlan_tag = <1 0x20>;  
};  
gmac1 {  
    local-mac-address = [000000000000];  
    vlan_tag = <2 0x1e>;  
};  
};
```

## ii) Linux Network Settings

File: /etc/config/network

```
config switch  
    option name 'switch0'  
    option reset '1'  
    option enable_vlan '1'
```

```
config switch_vlan  
    option device 'switch0'  
    option vlan '1'  
    option ports '0t 1 2 3 4'  
config switch_vlan  
    option device 'switch0'  
    option vlan '2'  
    option ports '0t 5'
```

## Test Steps

| No | Step Description   | Expected Results                 |
|----|--|----------------------------------|
| 1  | Check number of eth interfaces   | There should be 2, eth0 and eth1 |
| 2  | Connect one of the external spirent/exia ports to Port5(eth0) and Connect the other one to any one port from the remaining 4 ports(eth1) |                                  |
| 3  | Start bidi IPv4 UDP traffic using the traffic generator.   |                                  |
| 4  | Check Bidi Performance   |                                  |

### Pattern1 Traffic Type:

Spirent Port1 → Ipv4 UDP traffic through DUT → Spirent port 2

## 6.25.4 5 VLANs (1+1+1+1+1)

### Configuration Steps

#### 1. DTSI config

```
File: qcom-ipq40xx.dtsi
aliases {
    spi0 = &spi_0;
    spi1 = &spi_1;
    i2c0 = &i2c_0;
    i2c1 = &i2c_1;
    ethernet0 = "/soc/edma/gmac0";
    ethernet1 = "/soc/edma/gmac1";
    ethernet2 = "/soc/edma/gmac2";
    ethernet3 = "/soc/edma/gmac3";
    ethernet4 = "/soc/edma/gmac4";
};

edma@c080000 {
```

qcom,rx\_head\_buf\_size = <1540>;

qcom,num\_gmac = <5>;

gmac0 {

```
local-mac-address = [000000000000];
qcom,poll_required = <0x1>;
qcom,phy_mdio_addr = <4>;
qcom,forced_speed = <1000>;
qcom,forced_duplex = <1>;
vlan_tag = <2 0x20>;
};

gmac1 {
local-mac-address = [000000000000];
vlan_tag = <1 0x2>;
};

gmac2 {
local-mac-address = [000000000000];
vlan_tag = <3 0x4>;
};

gmac3 {
local-mac-address = [000000000000];
vlan_tag = <4 0x8>;
};

gmac4 {
local-mac-address = [000000000000];
vlan_tag = <5 0x10>;
};

};
```

## ii) Linux Network Settings

File: /etc/config/network

```
config switch
option name 'switch0'
option reset '1'
```

```

option enable_vlan '1'

config switch_vlan
    option device 'switch0'
    option vlan '1'
    option ports '0t 1'

config switch_vlan
    option device 'switch0'
    option vlan '2'
    option ports '0t 5'

config switch_vlan
    option device 'switch0'
    option vlan '3'
    option ports '0t 2'

config switch_vlan
    option device 'switch0'
    option vlan '4'
    option ports '0t 3'

config switch_vlan
    option device 'switch0'
    option vlan '5'
    option ports '0t 4'

```

## Test Steps

| No | Step Description  | Expected Results                             |
|----|---|--|
| 1  | Check number of eth interfaces  | There should be 5; eth0,eth1, eth2,eth3,eth4 |
| 2  | Connect one of the external spirent/exia ports to any eth interface and Connect the other one to any other port |  |
| 3  | Start bidi IPv4 UDP traffic using the traffic generator.  |  |
| 4  | Check Bidi Performance  |  |

**Pattern1 Traffic Type:**

Spirent Port1 → IPv4 UDP traffic through DUT → Spirent Port2

### 6.25.5 2 VLANs (3 + 2)

Configuration Steps

1. DTSI config

File: qcom-ipq40xx.dtsi

```
aliases {  
    spi0 = &spi_0;  
    spi1 = &spi_1;  
    i2c0 = &i2c_0;  
    i2c1 = &i2c_1;  
    ethernet0 = "/soc/edma/gmac0";  
    ethernet1 = "/soc/edma/gmac1";  
};  
edma@c080000 {  
  
    qcom,rx_head_buf_size = <1540>;  
    qcom,num_gmac = <2>;  
    gmac0 {  
        local-mac-address = [000000000000];  
        qcom,poll_required = <0x1>;  
        qcom,phy_mdio_addr = <4>;  
        qcom,forced_speed = <1000>;  
        qcom,forced_duplex = <1>;  
        vlan_tag = <2 0x30>;  
    };  
    gmac1 {  
        local-mac-address = [000000000000];  
        vlan_tag = <1 0xe>;  
    };
```

```
};
```

ii) Linux Network Settings

File: /etc/config/network

```
config switch
    option name 'switch0'
    option reset '1'
    option enable_vlan '1'
```

```
config switch_vlan
    option device 'switch0'
    option vlan '1'
    option ports '0t 1 2 3'
config switch_vlan
    option device 'switch0'
    option vlan '2'
    option ports '0t 4 5'
```

## Test Steps

| No | Step Description  | Expected Results             |
|----|---|------------------------------|
| 1  | Check number of eth interfaces  | There should be 2; eth0,eth1 |
| 2  | Connect one of the external spirent/exia port to port4/port5(eth0) and Connect the other one to port1/port2/port3(eth1) |                              |
| 3  | Start bidi IPv4 UDP traffic using the traffic generator.  |                              |
| 4  | Check Bidi Performance  |                              |

### Pattern1 Traffic Type:

Spirent Port1 → IPv4 UDP traffic through DUT → Spirent Port2

## **6.25.6 3 VLANs (2 + 2 + 1)**

## Configuration Steps

## 1. DTSI config

File: qcom-ipq40xx.dtsi

```
aliases {  
    spi0 = &spi_0;  
    spi1 = &spi_1;  
    i2c0 = &i2c_0;  
    i2c1 = &i2c_1;  
    ethernet0 = "/soc/edma/gmac0";  
    ethernet1 = "/soc/edma/gmac1";  
    ethernet2 = "/soc/edma/gmac2";  
};  
edma@c080000 {  
    .  
    .  
    .  
    .  
    .  
    .  
    .  
    qcom,rx_head_buf_size = <1540>;  
    qcom,num_gmac = <3>;  
    gmac0 {  
        local-mac-address = [000000000000];  
        qcom,poll_required = <0x1>;  
        qcom,phy_mdio_addr = <4>;  
        qcom,forced_speed = <1000>;  
        qcom,forced_duplex = <1>;  
        vlan_tag = <2 0x20>;  
    };  
    gmac1 {  
        local-mac-address = [000000000000];  
        vlan_tag = <1 0x6>;  
    };  
    gmac2 {  
        local-mac-address = [000000000000];
```

```
vlan_tag = <3 0x18>;
};
```

```
};
```

## ii) Linux Network Settings

File: /etc/config/network

```
config switch
    option name 'switch0'
    option reset '1'
    option enable_vlan '1'
```

```
config switch_vlan
    option device 'switch0'
    option vlan '1'
    option ports '0t 1 2'
config switch_vlan
    option device 'switch0'
    option vlan '2'
    option ports '0t 5'
config switch_vlan
    option device 'switch0'
    option vlan '3'
    option ports '0t 3 4'
```

## Test Steps: Combination 1

| No | Step Description   | Expected Results                  |
|----|--|-----------------------------------|
| 1  | Check number of eth interfaces   | There should be 3; eth0,eth1,eth2 |
| 2  | Connect one of the external spirent/exia port to port5(eth0) and Connect the other one to port1/port2 (eth1) |                                   |
| 3  | Start bidi IPv4 UDP traffic using the traffic generator.   |                                   |
| 4  | Check Bidi Performance   |                                   |

### 6.25.7 Commands to check SFE counters

```
#!/bin/sh
#@sfe_dump
#@example : sfe_dump
sfe_dump() {
[ -e "/dev/sfe_ipv4" ] || {
dev_num=$(cat /sys/sfe_ipv4/debug_dev)
mknod /dev/sfe_ipv4 c $dev_num 0
}
[ -e "/dev/sfe_ipv6" ] || {
dev_num=$(cat /sys/sfe_ipv6/debug_dev)
mknod /dev/sfe_ipv6 c $dev_num 0
}
cat /dev/sfe_$1
}
sfe_dump ipv4
sfe_dump ipv6
```

### 6.25.8 Sample output of ECM dump

**NOTE** 186 indicates the flow number and individual parameters are self-explanatory.

```
conns.conn.186.serial=186
conns.conn.186.sip_address=5aaa:0000:0000:0000:0000:0000:0000:0002
conns.conn.186.sip_address_
nat=5aaa:0000:0000:0000:0000:0000:0000:0002
conns.conn.186.sport=63
conns.conn.186.sport_nat=63
conns.conn.186.snode_address=00:00:00:8c:69:9d
conns.conn.186.snode_address_nat=00:00:00:8c:69:9d
conns.conn.186.dip_address=ff38:0000:0000:0000:0000:8000:000b
conns.conn.186.dip_address_
nat=ff38:0000:0000:0000:0000:8000:000b
conns.conn.186.dport=63
conns.conn.186.dport_nat=63
conns.conn.186.dnode_address=33:33:80:00:00:0b
conns.conn.186.dnode_address_nat=33:33:80:00:00:0b
conns.conn.186.ip_version=6
conns.conn.186.protocol=17
conns.conn.186.is_routed=1
conns.conn.186.expires=300
conns.conn.186.direction=2
conns.conn.186.time_added=5547
```

```
conns.conn.186.regen_success=0
conns.conn.186.regen_fail=0
conns.conn.186.regen_required=0
conns.conn.186.regen_occurrences=0
conns.conn.186.regen_in_progress=0
conns.conn.186.generation=0/0
conns.conn.186.adv_stats.from_data_total=414210000
conns.conn.186.adv_stats.to_data_total=0
conns.conn.186.adv_stats.from_packet_total=276140
conns.conn.186.adv_stats.to_packet_total=0
conns.conn.186.adv_stats.from_data_total_dropped=0
conns.conn.186.adv_stats.to_data_total_dropped=0
conns.conn.186.adv_stats.from_packet_total_dropped=0
conns.conn.186.adv_stats.to_packet_total_dropped=0
conns.conn.186.from_interfaces.interface_count=1
conns.conn.186.from_interfaces.0.ethernet iface.adv_stats.from_data_
total=0
conns.conn.186.from_interfaces.0.ethernet iface.adv_stats.to_data_
total=6627210000
conns.conn.186.from_interfaces.0.ethernet iface.adv_stats.from_
packet_total=0
conns.conn.186.from_interfaces.0.ethernet iface.adv_stats.to_packet_
total=4418140
conns.conn.186.from_interfaces.0.ethernet iface.adv_stats.from_data_
total_dropped=0
conns.conn.186.from_interfaces.0.ethernet iface.adv_stats.to_data_
total_dropped=0
conns.conn.186.from_interfaces.0.ethernet iface.adv_stats.from_
packet_total_dropped=0
conns.conn.186.from_interfaces.0.ethernet iface.adv_stats.to_packet_
total_dropped=0
conns.conn.186.from_interfaces.0.ethernet iface.type=0
conns.conn.186.from_interfaces.0.ethernet iface.name=eth0
conns.conn.186.from_interfaces.0.ethernet iface.time_added=5547
conns.conn.186.from_interfaces.0.ethernet iface.mtu=1500
conns.conn.186.from_interfaces.0.ethernet iface.interface_
identifier=2
conns.conn.186.from_interfaces.0.ethernet iface.ae_interface_
identifier=0
conns.conn.186.from_interfaces.0.ethernet iface.nodes=1
conns.conn.186.from_interfaces.0.ethernet address=00:03:7f:23:ed:96
conns.conn.186.to_mc_interfaces.interface_count=2
conns.conn.186.to_mc_interfaces.0.ethernet iface.adv_stats.from_data_
total=0
```

```
conns.conn.186.to_mc_interfaces.0.ethernet iface.adv_stats.to_data_
total=0
conns.conn.186.to_mc_interfaces.0.ethernet iface.adv_stats.from_
packet_total=0
conns.conn.186.to_mc_interfaces.0.ethernet iface.adv_stats.to_packet_
total=0
conns.conn.186.to_mc_interfaces.0.ethernet iface.adv_stats.from_data_
total_dropped=0
conns.conn.186.to_mc_interfaces.0.ethernet iface.adv_stats.to_data_
total_dropped=0
conns.conn.186.to_mc_interfaces.0.ethernet iface.adv_stats.from_
packet_total_dropped=0
conns.conn.186.to_mc_interfaces.0.ethernet iface.adv_stats.to_packet_
total_dropped=0
conns.conn.186.to_mc_interfaces.0.ethernet iface.type=0
conns.conn.186.to_mc_interfaces.0.ethernet iface.name=eth2
conns.conn.186.to_mc_interfaces.0.ethernet iface.time_added=5547
conns.conn.186.to_mc_interfaces.0.ethernet iface.mtu=1500
conns.conn.186.to_mc_interfaces.0.ethernet iface.interface_
identifier=4
conns.conn.186.to_mc_interfaces.0.ethernet iface.ae_interface_
identifier=2
conns.conn.186.to_mc_interfaces.0.ethernet iface.nodes=0
conns.conn.186.to_mc_interfaces.0.ethernet address=00:03:7f:5b:84:f5
conns.conn.186.to_mc_interfaces.1.bridge iface.adv_stats.from_data_
total=0
conns.conn.186.to_mc_interfaces.1.bridge iface.adv_stats.to_data_
total=6626692500
conns.conn.186.to_mc_interfaces.1.bridge iface.adv_stats.from_packet_
total=0
conns.conn.186.to_mc_interfaces.1.bridge iface.adv_stats.to_packet_
total=4417795
conns.conn.186.to_mc_interfaces.1.bridge iface.adv_stats.from_data_
total_dropped=0
conns.conn.186.to_mc_interfaces.1.bridge iface.adv_stats.to_data_
total_dropped=0
conns.conn.186.to_mc_interfaces.1.bridge iface.adv_stats.from_packet_
total_dropped=0
conns.conn.186.to_mc_interfaces.1.bridge iface.adv_stats.to_packet_
total_dropped=0
conns.conn.186.to_mc_interfaces.1.bridge iface.type=4
conns.conn.186.to_mc_interfaces.1.bridge iface.name=br-lan
conns.conn.186.to_mc_interfaces.1.bridge iface.time_added=5547
conns.conn.186.to_mc_interfaces.1.bridge iface.mtu=1500
```

```
conns.conn.186.to_mc_interfaces.1.bridge iface.interface_
identifier=16
conns.conn.186.to_mc_interfaces.1.bridge iface.ae_interface_
identifier=-1
conns.conn.186.to_mc_interfaces.1.bridge iface.nodes=16
conns.conn.186.to_mc_interfaces.1.bridge address=00:03:7f:4b:81:39
conns.conn.186.from_nat_interfaces.interface_count=0
conns.conn.186.to_nat_interfaces.interface_count=0
conns.conn.186.front_end_v6.multicast.can_accel=1
conns.conn.186.front_end_v6.multicast.accel_mode=2
conns.conn.186.front_end_v6.multicast.decelerate_pending=0
conns.conn.186.front_end_v6.multicast.flush_happened_total=0
conns.conn.186.front_end_v6.multicast.no_action_seen_total=0
conns.conn.186.front_end_v6.multicast.no_action_seen=0
conns.conn.186.front_end_v6.multicast.no_action_seen_limit=250
conns.conn.186.front_end_v6.multicast.driver_fail_total=0
conns.conn.186.front_end_v6.multicast.driver_fail=0
conns.conn.186.front_end_v6.multicast.driver_fail_limit=250
conns.conn.186.front_end_v6.multicast.ae_nack_total=0
conns.conn.186.front_end_v6.multicast.ae_nack=0
conns.conn.186.front_end_v6.multicast.ae_nack_limit=250
conns.conn.186.classifiers.default.ingress_sender=1
conns.conn.186.classifiers.default.egress_sender=0
conns.conn.186.classifiers.default.timer_group=3
conns.conn.186.classifiers.default.pr.relevant=yes
conns.conn.186.classifiers.default.pr.accel=wanted
conns.conn.186.classifiers.default.pr.timer_group=3
conns.conn.186.classifiers.default.trackers.tracker_udp.src_count=0
conns.conn.186.classifiers.default.trackers.tracker_udp.src_bytes_
total=0
conns.conn.186.classifiers.default.trackers.tracker_udp.dest_count=0
conns.conn.186.classifiers.default.trackers.tracker_udp.dest_bytes_
total=0
conns.conn.186.classifiers.default.trackers.tracker_udp.data_
limit=1048576
conns.conn.186.classifiers.default.trackers.tracker_udp.timer_group=1
conns.conn.186.classifiers.default.trackers.tracker_udp.src_sender_
state=Established
conns.conn.186.classifiers.default.trackers.tracker_udp.dest_sender_
state=Unknown
conns.conn.186.classifiers.default.trackers.tracker_udp.connection_
state=Established
conns.conn.186.classifiers.hyfi.pr.relevant=yes
conns.conn.186.classifiers.dscp.pr.relevant=yes
```

```
conns.conn.186.classifiers.dscp.pr.flow_qos_tag=0
conns.conn.186.classifiers.dscp.pr.return_qos_tag=0
```

### 6.25.9 Sample output of SFE dump

```
<sfe_ipv4>
    <connections>
        <connection protocol="6" src_dev="br-lan" src_ip="192.168.1.22" src_ip_xlate="192.168.2.1" src_port="30048" src_port_xlate="30048" src_priority="0" src_dscp="0" src_rx_pkts="58172" src_rx_bytes="78648544" dest_dev="eth0" dest_ip="192.168.2.22" dest_ip_xlate="192.168.2.22" dest_port="5142" dest_port_xlate="5142" dest_priority="0" dest_dscp="0" dest_rx_pkts="57750" dest_rx_bytes="78078000" src_flow_cookie="0" dst_flow_cookie="0" last_sync="18" mark="bf8c8030" />
    </connections>
    <exceptions>
        <exception name="UDP_NO_CONNECTION" count="80143" />
        <exception name="TCP_NO_CONNECTION_SLOW_FLAGS" count="31513" />
        <exception name="TCP_NO_CONNECTION_FAST_FLAGS" count="305" />
        <exception name="TCP_FLAGS" count="100" />
        <exception name="UNHANDLED_PROTOCOL" count="166" />
    </exceptions>
    <stats num_connections="3" pkts_forwarded="89712845" pkts_not_forwarded="112244" create_requests="207" create_collisions="0" destroy_requests="104" destroy_misses="0" flushes="204" hash_hits="85002787" hash_reorders="4710158" />
</sfe_ipv4>
<sfe_ipv6>
    <connections>
    </connections>
    <exceptions>
        <exception name="ICMP_UNHANDLED_TYPE" count="659" />
    </exceptions>
    <stats num_connections="0" pkts_forwarded="0" pkts_not_forwarded="659" create_requests="0" create_collisions="0" destroy_requests="0" destroy_misses="0" flushes="0" hash_hits="0" hash_reorders="0" />
</sfe_ipv6>
```

### 6.25.10 Routing and multiflow configuration commands

#### Routing Configuration

```
uci set network.lan=interface
uci set network.lan.ifname='eth1'
uci set network.lan.proto=static
uci set network.lan.ipaddr=192.168.1.1
```

```
uci set network.lan.ip6addr=4aaa::1/64
uci set network.lan.netmask=255.255.255.0
uci set network.wan=interface
uci set network.wan.ifname=eth0
uci set network.wan.proto=static
uci set network.wan.ipaddr=172.16.10.10
uci set network.wan.ip6addr=5aaa::1/64
uci set network.wan.netmask=255.255.0.0
uci commit network
/etc/init.d/network restart
```

## Firewall settings

```
uci set firewall.@defaults[0]=defaults
uci set firewall.@defaults[0].syn_flood=1
uci set firewall.@defaults[0].input=ACCEPT
uci set firewall.@defaults[0].output=ACCEPT
uci set firewall.@defaults[0].forward=ACCEPT
uci set firewall.@zone[0]=zone
uci set firewall.@zone[0].name=lan
uci set firewall.@zone[0].network=lan
uci set firewall.@zone[0].input=ACCEPT
uci set firewall.@zone[0].output=ACCEPT
uci set firewall.@zone[0].forward=ACCEPT
uci set firewall.@zone[1]=zone
uci set firewall.@zone[1].name=wan
uci set firewall.@zone[1].network=wan
uci set firewall.@zone[1].input=ACCEPT
uci set firewall.@zone[1].output=ACCEPT
uci set firewall.@zone[1].forward=ACCEPT
uci set firewall.@zone[1].masq=1
uci set firewall.@zone[1].mtu_fix=1
uci commit firewall
/etc/init.d/firewall restart
```

## Streamboost configuration

**SB ON:**

```
uci set appflow.tccontroller.uplimit=143000000
uci set appflow.tccontroller.downlimit=143000000
uci set appflow.tccontroller.enable_streamboost=1
uci commit
    luci-reload appflow
```

**SB OFF:**

```
uci set appflow.tccontroller.enable_streamboost=0
uci commit
    luci-reload appflow
```

## DNAT Rule for Flows

Add following configuration in /etc/config/firewall.

```
config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.20'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.22'
```

```
config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.21'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.23'
```

```
config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.22'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.24'

config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.23'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.25'

config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.24'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.26'

config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
```

```
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.25'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.27'

config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.26'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.28'

config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.27'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.29'

config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.28'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.30'
```

```
config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.29'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.31'
```

```
config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.30'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.32'
```

```
config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.31'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.33'
```

```
config redirect
option target 'DNAT'
option src 'wan'
```

```
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.32'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.34'

config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.33'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.35'

config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.34'
option src_dip '172.16.10.10'
option dest_ip '192.168.1.36'

config redirect
option target 'DNAT'
option src 'wan'
option dest 'lan'
option proto 'tcp udp'
option name 'DNAT'
option src_ip '172.16.10.35'
option src_dip '172.16.10.10'
```

```
option dest_ip '192.168.1.37'
```

### 6.25.11 Commands to disable and enable SFE

#### Manually load SFE:

```
insmod ecm  
insmod shortcut_fe_drv  
insmod shortcut_fe_ipv6
```

#### Manually unload SFE

```
rmmod ecm  
rmmod shortcut_fe_drv  
rmmod shortcut_fe_ipv6
```

### 6.25.12 HNAT commands

#### ENABLE HNAT

```
ssdk_sh nat global set enable disable
```

#### DYNAMIC HNAT

```
iptables -A POSTROUTING -t nat -o eth0 -j MASQUERADE
```

#### STATIC HNAT

```
iptables -t nat -A POSTROUTING -p udp -s 192.168.1.10 -d 192.168.2.10 -j SNAT --to 192.168.2.1
```

### 6.25.13 HNAT counters

```
ssdk_sh nat naptentry show
```

### 6.25.14 ECM counters

```
cat /sys/kernel/debug/ecm/ecm_db/connection_count  
cat /sys/kernel/debug/ecm/ecm_db/connection_count_simple  
cat /sys/kernel/debug/ecm/ecm_db/iface_count
```

```
cat /sys/kernel/debug/ecm/ecm_sfe_ipv6/udp_accelerated_count  
cat /sys/kernel/debug/ecm/ecm_sfe_ipv4/udp_accelerated_count  
cat /sys/kernel/debug/ecm/ecm_sfe_ipv6/tcp_accelerated_count  
cat /sys/kernel/debug/ecm/ecm_sfe_ipv4/tcp_accelerated_count
```

## 6.26 ACL-based steering

This chapter describes how to use ACL and L3 ARP table to implement flow steering for GRE tunnel traffic for IPQ4018/IPQ4019/IPQ4028/IPQ4029 chipsets. It also explains the feature and functions in IPQ4018/IPQ4019/IPQ4028/IPQ4029 ACL-based steering from the software aspect according to the software requirement specification. It gives a guidance on how to set up ACL-based flow steering.

### 6.26.1 ESS switch ACL feature

The ESS supports up to 96 ACL rule table entries. Each rule can support filtering or redirection of the incoming packets based on the following field in the packet.

- Layer 2
  - Source MAC address
  - Destination MAC address
  - VLAN ID
  - EtherType
- Layer 3
  - Source IP address
  - Destination IP address
  - Protocol
- Layer 4
  - Source TCP/UDP port number
  - Destination TCP/UDP port number
- Binding port
  - Physical port number
- Window pattern
  - User-defined window pattern

When the incoming packets match an entry in the rule table, the following action can be taken defined in the result field.

- Change C-Tag or S-Tag
- Drop or redirect the packet

- Configure rate limit based on flow
- Change priority
- Force layer 3 and NAT operation

The ESS can bind multiple keys and support up to two matches per packet to implement functions such as QoS, forwarding, routing, and rate measuring/limiting.

Especially for windows rule, total five windows can be enabled simultaneously based on one port.

- L2
- L2Plus
- L3
- L3plus
- L4

For every window, offset and length can be configured to define the window.

- If length is 0, this window is disabled. The packet bytes are fetched according to these five windows.
- If there is overlap between two windows, duplicate bytes are not fetched. For example, if byte3, byte4, byte5 are fetched from window1, and byte4, byte5, byte6 are fetched from window2, then the packet bytes to be used to compare are byte3, byte4, byte5, byte6, instead of byte3, byte4, byte5, byte4, byte5, byte6.

The following figure shows the fetched bytes from the packet, and the per port offset/length configuration.

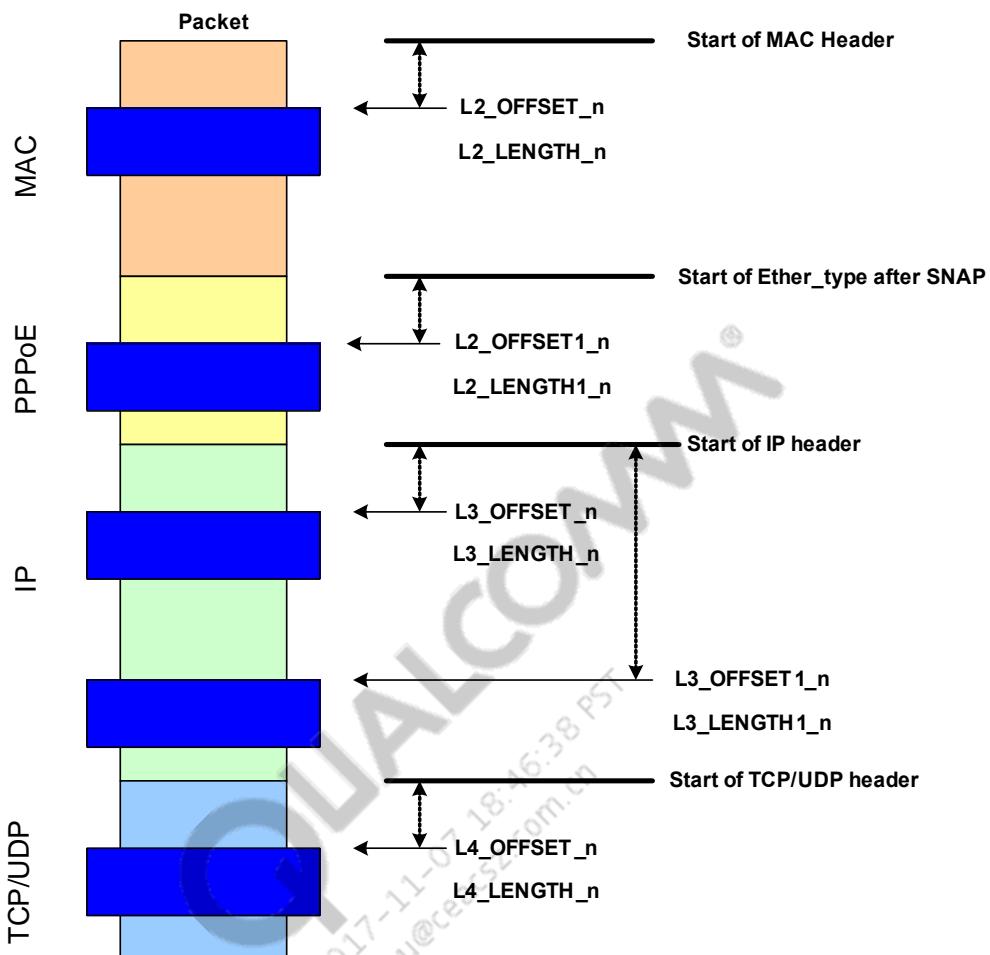


Figure 6-25 ACL window rule

- If more than 16 bytes are got from the offset, the first 16 bytes are fetched to compare with the ACL window pattern.
- Other procedures are the same as other pattern.

### 6.26.2 GRE flow steering use case

In the ESS design, the idea to implement accelerated flow steering is to identify the flow by looking up a table, and an output value named load balance value is returned. The load balance value is used to map to a specific CPU ring by a configurable mapping table in the EDMA. Each ring can be finally bound to different CPU core via the independent interrupts.

The purpose of GRE flow steering is to steer the GRE tunnel related packets to different CPU core, to reduce the cache missed rate. On IPQ4018/IPQ4019/IPQ4028/IPQ4029 platform, the mapping of load balance value to CPU core is shown in the following table.

Table 6-8 Mapping between load balance value and CPU core

| Load balance value | CPU core# |
|--------------------|-----------|
| 0                  | 0         |
| 1                  | 1         |
| 2                  | 2         |
| 3                  | 3         |

As shown in the preceding table, different load balance values are used to perform traffic steering.

Based on GRE tunnel's requirement that GRE tunnel packet must be steered to different CPU core based on tunnel ID, different GRE tunnel ID is redirected to expected load balance value.

- To add load balance option into GRE tunnel packet, ARP entry must be used to generate given load balance value.
- To match ARP entry, route interface entry must be created to enter into L3 process of switch core firstly.
- To identify GRE tunnel packet and assign correlate ARP index, ACL WINDOWS rule function of switch core is used to implement it.

### 6.26.3 ARP entry with load balance

In IPQ4018/IPQ4019/IPQ4028/IPQ4029 switch core, ARP entry is provided to implement some actions such as load balance for matched packet. In this case, one reserved ARP entry is used and expected load balance action is the key. Total four load balance values are used for four CPU cores. So four IPv4 ARP entries and four IPv6 ARP entries must be reserved to generate four load balance values.

```
/*Reserved IPv4 0.0.0.1 is used to map to CPU core 0*/
ssdk_sh ip hostentry add 0 1 7 0.0.0.1 00-00-00-00-00-00 1 4 0 0 rdtcpu n
y 0
/*Reserved IPv4 0.0.0.2 is used to map to CPU core 1*/
ssdk_sh ip hostentry add 0 1 7 0.0.0.2 00-00-00-00-00-00 1 5 0 0 rdtcpu n
y 0
/*Reserved IPv4 0.0.0.3 is used to map to CPU core 2*/
ssdk_sh ip hostentry add 0 1 7 0.0.0.3 00-00-00-00-00-00 1 6 0 0 rdtcpu n
y 0
/*Reserved IPv4 0.0.0.4 is used to map to CPU core 3*/
ssdk_sh ip hostentry add 0 1 7 0.0.0.4 00-00-00-00-00-00 1 7 0 0 rdtcpu n
y 0
/*Reserved IPv6 1::9 is used to map to CPU core 0*/
ssdk_sh ip hostentry add 0 2 7 1::9 00-00-00-00-00-00 1 4 0 0 rdtcpu n y 1
/*Reserved IPv6 2::9 is used to map to CPU core 1*/
ssdk_sh ip hostentry add 0 2 7 2::9 00-00-00-00-00-00 1 5 0 0 rdtcpu n y 1
/*Reserved IPv6 3::9 is used to map to CPU core 2*/
ssdk_sh ip hostentry add 0 2 7 3::9 00-00-00-00-00-00 1 6 0 0 rdtcpu n y 1
/*Reserved IPv6 4::9 is used to map to CPU core 3*/
ssdk_sh ip hostentry add 0 2 7 4::9 00-00-00-00-00-00 1 7 0 0 rdtcpu n y 1
```

After these ARP entries are created, the related ARP entries' index can be retrieved:

```
[entryid]:0x0 [ip_addr]:0.0.0.3
[entryid]:0x1 [ip_addr]:2:0:0:0:0:0:0:9
[entryid]:0x20 [ip_addr]:0.0.0.2
```

```
[entryid]:0x21 [ip_addr]:1:0:0:0:0:0:0:0:9
[entryid]:0x40 [ip_addr]:0.0.0.1
[entryid]:0x48 [ip_addr]:4:0:0:0:0:0:0:0:9
[entryid]:0x68 [ip_addr]:0.0.0.4
[entryid]:0x78 [ip_addr]:3:0:0:0:0:0:0:0:9
```

These indexes are used by ACL ARP action.

#### 6.26.4 Interface entry with MAC

To ensure that Layer 3 process can be entered into, one interface entry must be set up. This interface entry matches the destination MAC address to decide whether L3 process can be involved.

This interface entry must be created before ARP entry creation.

```
/*This destination MAC address should be DUT Ethernet Interface MAC
address*/
ssdk_sh ip intfentry add 0 0 1 1 EA-48-5B-D7-43-4C y y
```

**NOTE** This interface entry is created when Linux Ethernet interfaces are created probably. So it may be not necessary.

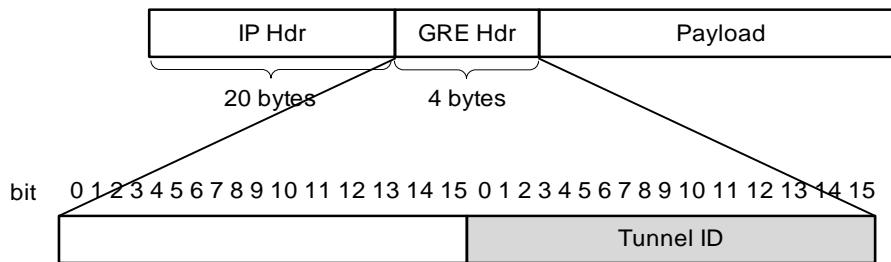
#### 6.26.5 GRE tunnel identification

Two main properties can be used to identify one GRE tunnel packet with specific tunnel ID:

- IP protocol is 0x2f
- Tunnel ID

The switch core can identify IP protocol, and only the window ACL rule can identify specific GRE tunnel ID.

For IPv4 GRE tunnel packet, the tunnel ID is transported in 2 bytes of the GRE header in network byte order and is key to identify the offload candidate flows (see the following figure).



**Figure 6-26 IPv4 GRE tunnel packet**

As shown in the preceding figure, the GRE tunnel ID is in the 23<sup>rd</sup> and 24<sup>th</sup> bytes from the L3 header, so the windows offset value is 22 and 2 bytes are matched.

For IPv6 GRE tunnel packet, the IP header is 40 bytes. The maximum offset of window is 31, so more bytes must be taken for checking. The window length is 13.

When the GRE tunnel packet is matched, L3 action is performed with reserved ARP entries index to decide which load balance or which CPU core is selected.

In this way, traffic with one special GRE tunnel ID can be steered to expected CPU core, and ESS ACL module can support mask of checking field, so more traffic with one more special GRE tunnel ID can be also steered into one same expected CPU core.

The following shows related ACL configurations:

- ACL UDF profile configuration

```
ssdk_sh acl udfprofile set 1 13 22 2
ssdk_sh acl udfprofile set 1 13plus 31 13
```

- ACL rule to identify GRE with tunnel ID 0x44 packet and map to CPU core 0

```
//ipv4 rule for GRE(0x2f) and tunnel id 0x44 mapping internal
priority //as 0 with arp action 0x60
ssdk_sh acl rule add 10 0 1 ip4 n n n n n n n n n n n n n n n n n n n n n n
n y 0x2f 0xff n n n n n n y 12 0 00-44 ff-ff n y n n n n n n n y 0
n n 0 0 n n n n n n n n y 0x40 n n n
//ipv6 rule for GRE(0x2f) and tunnel id 0x44 mapping internal
priority //as 0 with arp action 0x20
ssdk_sh acl rule add 10 1 1 ip6 n n n n n n n n n n n n n n n n n n n n n
n y 0x2f 0xff n n n n n n y 12 0 00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-44 00-00-00-00-00-00-00-00-00-00-00-00-ff-ff n y n n n
n n n n y 0 n n 0 0 n n n n n n n n y 0x21 n n n
```

- ACL other configuration

```
ssdk_sh acl status set enable
ssdk_sh acl list create 10 10
ssdk_sh acl list bind 10 0 0 1
```

## 6.26.6 UCI configuration

All switch core configurations can be implemented by UCI.

### 6.26.6.1 ACL related UCI configuration

There are two parts for ACL module:

- UDF profile part
- ACL rule part

#### UDF profile part

The following table shows the configuration format.

**Table 6-9 UDF profile configuration format**

| UCI format   | Description  |
|--|--|
| <pre>config switch_ext     option device 'switch0'     option name 'AclUdfprofile'     option port '2'     option user_defined_type 'l2'     option user_defined_offset '1'     option user_defined_length '4'</pre> | <p>Set user defined field offset and length on a particular port.</p> <ul style="list-style-type: none"> <li>§ port = 0-6</li> <li>§ user_defined_type = l2/l2snap/l3/l3plus/l4</li> <li>§ user_defined_offset = offset value</li> <li>§ user_defined_length = 0-15</li> </ul> |

In this case, the UCI profile configuration is as follows:

```
config switch_ext
    option device 'switch0'
    option name 'AclUdfprofile'
    option port '1'
    option user_defined_type 'l3'
    option user_defined_offset '22'
    option user_defined_length '2'

config switch_ext
    option device 'switch0'
    option name 'AclUdfprofile'
    option port '1'
    option user_defined_type 'l3plus'
    option user_defined_offset '31'
    option user_defined_length '13'
```

### ACL rule part

**Table 6-10 ACL rule confirmation format**

| UCI format  | Description   |
|---|---|
| <pre> config switch_ext /*MUST have field for one rule*/     option device 'switch0'     option name 'AclRule'     option rule_id '1'     option priority '1'     option rule_type 'ip4'     option port_bitmap '0x1e'  /*Basic L2 checking field*/     option dst_mac_address '00-00-00-00-00-00'     option dst_mac_address_mask 'ff-ff-ff-ff-ff-ff'     option src_mac_address '00-00-00-00-00-00'     option src_mac_address_mask 'ff-ff-ff-ff-ff-ff'     option ethernet_type '3'     option ethernet_type_mask '0xff'     option vlan_id '3'     option vlan_id_mask '0xff'     option vlan_priority '1'     option vlan_priority_mask '1'     option tagged '1'     option tagged_mask '1'     option cfi '1'  /*STAG mode, enhanced L2 checking field*/     option ctagged '1'     option ctag_vlan_id '3'     option ctag_vlan_id_mask '0xff'     option ctag_vlan_priority '1'     option ctag_vlan_priority_mask '1'     option ctag_cfi '1'      option stagged '1'     option stag_vlan_id '3'     option stag_vlan_id_mask '0xff'     option stag_vlan_priority '1'     option stag_vlan_priority_mask '1'     option stag_dei '1' </pre> | <pre> § rule_id = 1-95 § priority = 1-95 § rule_type = MAC   IPv4   IPv6   UDF § dst_mac_address = destination MAC § dst_mac_address_mask = destination MAC mask § src_mac_address = source MAC § src_mac_address_mask = source MAC mask § ethernet_type = 0-0xffff § vlan_id = 0-4095 § vlan_priority = 0-7 § tagged = 0   1 § cfi = 0 1 § ctag_vlan_id = 0-4095 § ctag_vlan_priority = 0-7 § ctagged = 0   1 § ctag_cfi = 0   1 § stag_vlan_id = 0-4095 § stag_vlan_priority = 0-7 § staged = 0   1 § stag_cfi = 0   1 § ipv4_src_address = IPv4 source IP address § ipv4_dst_address = IPv4 destination IP § ipv6_src_address = IPv4 source IP address § ipv6_dst_address = IPv4 destination IP § ipv6_flow_label = IPv6 flow label § ip_protocol = 0-0xff § ip_dscp = 0-63 § ip_dst_port = IP destination port § ip_src_port = IP source port § icmp_type = ICMP type § icmp_code = ICMP code § tcp_flag = TCP flag § ripv1 = RIP packet § dhcpcv4 = DHCPCv4 packet § dhcpcv6 = DHCPCv6 packet § inverse_check_fields = 0   1     ú 0 = Do not inverse any field.     ú 1 = Inverse all fields in this rule. </pre> |

|  |  |
|--|--|
| <pre> /*IPv4 L3 checking field*/ option ipv4_src_address '1.1.1.1' option ipv4_src_address_mask '1.1.1.1' option ipv4_dst_address '1.1.1.1' option ipv4_dst_address_mask '1.1.1.1'  /*IPv6 L3 checking field*/ option ipv6_src_address 'ff::00' option ipv6_src_address_mask 'ff::00' option ipv6_dst_address 'ff::00' option ipv6_dst_address_mask 'ff::00' option ipv6_flow_label '0x12345' option ipv6_flow_label_mask '0xfffff'  /*IP L4 checking field*/ option ip_protocol '1' option ip_protocol_mask '1' option ip_dscp '1' option ip_dscp_mask '1' option ip_dst_port '3' option ip_dst_port_mask '0xff' option ip_src_port '3' option ip_src_port_mask '0xff'  /*ICMP tcp rip dhcp flag checking field*/ option icmp_type '100' option icmp_type_mask '0xff' option icmp_code '100' option icmp_code_mask '0xff' option tcp_flag '100' option tcp_flag_mask '0xff'  /*RIP, DHCP checking field*/ option ripv1 '1' option dhcpv4 '1' option dhcpv6 '1' </pre> | <p>§ packet_drop = yes   no<br/>     § redirect_to_cpu = yes   no<br/>     § copy_to_cpu = yes   no<br/>     § redirect_to_ports = destination ports bitmap<br/>     § mirror = yes   no<br/>     § dscp_of_remark = 0-63<br/>     § queue_of_remark = 0-7<br/>     § port_bitmap = define into which ports this rule binds.<br/>         ú Bit[0] for port 0<br/>         ú Bit[1] for port 1<br/>         ú ...<br/>         ú Bit[6] for port 6<br/>     § remark_lookup_vid = yes   no<br/>     § stag_vid_of_remark = 0-4095<br/>     § stag_priority_of_remark = 0-7<br/>     § stag_dei_of_remark = 0   1<br/>     § ctag_vid_of_remark = 0-4095<br/>     § ctag_priority_of_remark = 0-7<br/>     § ctag_dei_of_remark = 0   1<br/>     § action_policer_id = 0-31<br/>     § action_arp_ptr = 0-127<br/>     § action_wcmp_ptr = 0-3<br/>     § action_snat = yes   no<br/>     § action_dnat = yes   no<br/>     § bypass_egress_translation = yes   no<br/>     § interrupt_trigger = yes   no   </p> |
|--|--|

```

/*rule forward action. If drop is yes, no action is valid.
redirect_to_cpu and copy_to_cpu can't exist together.*/
    option inverse_check_fields 'no'
    option packet_drop 'no'
    option redirect_to_cpu 'y'
    option copy_to_cpu 'y'
    option redirect_to_ports '0x1e'
    option mirror 'y'

/*rule vlan change action*/
    option remark_lookup_vid 'y'
    option stag_vid_of_remark '100'
    option stag_priority_of_remark '2'
    option stag_dei_of_remark '1'
    option ctag_vid_of_remark '100'
    option ctag_priority_of_remark '10'
    option ctag_dei_of_remark '4'

/*rule dscp change action*/
    option dscp_of_remark '1'
/*rule queue change action*/
    option queue_of_remark '1'

/*rule rate limit action*/
    option action_policer_id '1'

/*rule L3 action. arp and wcmp cannot exist together;
snat and dnat cannot exist together.*/
    option action_arp_ptr '1'
    option action_wcmp_ptr '1'
    option action_snat 'y'
    option action_dnat 'y'

/*rule egress qinq bypass action and interrupt trigger
action*/
    option bypass_egress_translation 'y'
    option interrupt_trigger 'y'

```

§

For ACL rule, only checked field must be considered, so the following ACL UCI can be used:

```

/*IPv4 rule*/
config switch_ext
    option device 'switch0'
    option name 'AclRule'
    option rule_id '10'
    option priority '10'
    option rule_type 'ip4'
    option ip_protocol '0x2f'

```

```

option ip_protocol_mask '0xff'
option user_defined_field_value '00-44'
option user_defined_field_mask 'ff-ff'
option action_arp_ptr '0x60'
option port_bitmap '0x2'

/*IPv6 rule*/
config switch_ext
    option device 'switch0'
    option name 'AclRule'
    option rule_id '20'
    option priority '20'
    option rule_type 'ip6'
    option ip_protocol '0x2f'
    option ip_protocol_mask '0xff'
    option user_defined_field_value '00-00-00-00-00-00-00-00-00-00-00-00-00-00-44'
    option user_defined_field_mask '00-00-00-00-00-00-00-00-00-00-00-00-00-ff-ff'
    option action_arp_ptr '0x20'
    option port_bitmap '0x2'

```

### 6.26.6.2 ARP related UCI configuration

There are two main configurations:

- ARP
- Interface entry

#### **ARP**

The following table shows the ARP configuration format.

**Table 6-11 ARP configuration format**

| UCI command | Description |
|-------------|-------------|
|-------------|-------------|

|  |   |
|--|---|
| <pre> config switch_ext     option device 'switch0'     option name 'IpHostentry'     option entry_id '0'     option entry_flags '1'     option entry_status '0xf'     option ip_addr '1.1.1.1'     option mac_addr '00-00-00-00-00-00'     option interface_id '0'     option load_balance_num '0'     option vrf_id '0'     option port_id '4'     option action 'forward'     option mirror 'no'     option counter 'no' </pre> | <p>Add a host entry with following parameter:</p> <ul style="list-style-type: none"> <li>§ entryid = 0-1023</li> <li>§ entryflags <ul style="list-style-type: none"> <li>ú #define FAL_IP_IP4_ADDR 0x1</li> <li>ú #define FAL_IP_IP6_ADDR 0x2</li> <li>ú #define FAL_IP_CPU_ADDR 0x4</li> </ul> </li> <li>§ entry_status = 0-15 <ul style="list-style-type: none"> <li>ú 0 = entry is empty.</li> <li>ú 1-7 = entry is dynamic and valid.</li> <li>ú 8-14 = entry is dynamic and valid, can be aged but cannot be changed by any other address.</li> <li>ú 15 = entry is static and cannot be aged or changed by hardware.</li> </ul> </li> <li>§ ip4 addr = IPv4 address</li> <li>§ mac addr = MAC address</li> <li>§ interface id = 0-4094</li> <li>§ load_balance_num = 0-3 (Only available in IPQ4018/IPQ4019/IPQ4028/IPQ4029 ESS)</li> <li>§ vrf_id = 0-7 (Only available in IPQ4018/IPQ4019/IPQ4028/IPQ4029 ESS)</li> <li>§ port id = 0-6</li> <li>§ action = forward</li> <li>§ mirror = yes   no</li> <li>§ counter = yes   no</li> </ul> |
|--|---|

### IP4 ARP entry

```

config switch_ext
    option device 'switch0'
    option name 'IpHostentry'
    option entry_id '0'
    option entry_flags '1'
    option entry_status '7'
    option ip_addr '0.0.0.0'
    option mac_addr '00-00-00-00-00-00'
    option interface_id '1'
    option load_balance_num '4'
    option vrf_id '0'
    option port_id '0'
    option action 'rdtcpu'
    option mirror 'no'
    option counter 'no'

```

### IPv6 ARP entry

```

config switch_ext
    option device 'switch0'
    option name 'IpHostentry'
    option entry_id '0'

```

```

option entry_flags '2'
option entry_status '7'
option ip_addr '1::9'
option mac_addr '00-00-00-00-00-00'
option interface_id '1'
option load_balance_num '4'
option vrf_id '0'
option port_id '0'
option action 'rdtcpu'
option mirror 'no'
option counter 'no'

```

### 6.26.6.3 Interface entry

The following table shows the interface entry configuration format.

**Table 6-12 Interface entry configuration format**

| UCI command  | Description  |
|--|--|
| config switch_ext       option device 'switch0'       option name 'IpIntfentry'       option entry_id '0'       option vrf_id '0'       option vlan_low '1'       option vlan_high '1'       option mac_addr '00-00-00-00-00-11'       option ipv4_route 'yes'       option ipv6_route 'yes' | Add interface entry.<br>§ entry_id = 0-7<br>§ vrf_id = 0-7 (available in IPQ4018/IPQ4019/IPQ4028/IPQ4029 ESS only) |

#### Interface entry UCI

```

config switch_ext
  option device 'switch0'
  option name 'IpIntfentry'
  option entry_id '0'
  option vrf_id '0'
  option vlan_low '1'
  option vlan_high '1'
  option mac_addr 'ea-48-5b-d7-43-4c'
  option ipv4_route 'yes'
  option ipv6_route 'yes'

```

## 6.26.7 Limitations

**GRE identification limitation**—Due to switch ACL engine hardware limitation, no IP header option can be supported. It means that GRE tunnel packet with IP header option will not be matched by the ACL engine, so no steering can take effect.

GRE tunnel packet can be fragmented. In this fragmented packet, no protocol can be matched and ACL rule also cannot match them, so in fragment case no steering can take effect.

**Reserved IP address limitation**—Customers can use any their expected reserved IP address to create load balance option, so reserved IP address is not available for normal traffic. Total four IPv4 ARP entries and four IPv6 ARP entries must be reserved in this application.

**ACL rule number limitation**—Two hardware ACL rules are required for IPv4 packet and for IPv6 packet respectively, so for one GRE tunnel ID, four hardware ACL rules are required.

In IPQ4018/IPQ4019/IPQ4028/IPQ4029 ESS module, up to 96 ACL rules are supported.

## 6.26.8 Configuration example

The following shows an example for mapping GRE traffic with tunnel ID 0x44 to CPU core 0. Detailed /etc/config/network file is as follows, run /etc/init.d/network restart, it can work for GRE traffic.

```
config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
    option ipaddr '127.0.0.1'
    option netmask '255.0.0.0'

config globals 'globals'
    option ula_prefix 'auto'

config interface 'lan'
    option ifname 'eth1'
    option type 'bridge'
    option proto 'static'
    option ipaddr '192.168.1.1'
    option netmask '255.255.255.0'
    option ip6assign '60'

config interface 'wan'
    option ifname 'eth0'
    option proto 'dhcp'

config interface 'wan6'
    option ifname '@wan'
    option proto 'dhcipv6'
```

```
config switch
    option name 'switch0'
    option reset '1'
    option enable_vlan '1'

config switch_vlan
    option device 'switch0'
    option vlan '1'
    option ports '0t 1 2 3 4'

config switch_vlan
    option device 'switch0'
    option vlan '2'
    option ports '0t 5'

config switch_ext
    option device 'switch0'
    option name 'IpIntfentry'
    option entry_id '0'
    option vrf_id '0'
    option vlan_low '1'
    option vlan_high '1'
    option mac_addr 'ea-48-5b-d7-43-4c'
    option ipv4_route 'yes'
    option ipv6_route 'yes'

config switch_ext
    option device 'switch0'
    option name 'IpHostentry'
    option entry_id '0'
    option entry_flags '1'
    option entry_status '7'
    option ip_addr '0.0.0.1'
    option mac_addr '00-00-00-00-00-00'
    option interface_id '1'
    option load_balance_num '4'
    option vrf_id '0'
    option port_id '0'
    option action 'rdtcpu'
    option mirror 'no'
    option counter 'no'

config switch_ext
```

```
option device 'switch0'
option name 'IpHostentry'
option entry_id '0'
option entry_flags '1'
option entry_status '7'
option ip_addr '0.0.0.2'
option mac_addr '00-00-00-00-00-00'
option interface_id '1'
option load_balance_num '4'
option vrf_id '0'
option port_id '0'
option action 'rdtcpu'
option mirror 'no'
option counter 'no'

config switch_ext
    option device 'switch0'
    option name 'IpHostentry'
    option entry_id '0'
    option entry_flags '1'
    option entry_status '7'
    option ip_addr '0.0.0.3'
    option mac_addr '00-00-00-00-00-00'
    option interface_id '1'
    option load_balance_num '4'
    option vrf_id '0'
    option port_id '0'
    option action 'rdtcpu'
    option mirror 'no'
    option counter 'no'

config switch_ext
    option device 'switch0'
    option name 'IpHostentry'
    option entry_id '0'
    option entry_flags '1'
    option entry_status '7'
    option ip_addr '0.0.0.4'
    option mac_addr '00-00-00-00-00-00'
    option interface_id '1'
    option load_balance_num '4'
    option vrf_id '0'
    option port_id '0'
    option action 'rdtcpu'
```

```
option mirror 'no'
option counter 'no'

config switch_ext
    option device 'switch0'
    option name 'IpHostentry'
    option entry_id '0'
    option entry_flags '2'
    option entry_status '7'
    option ip_addr '1::9'
    option mac_addr '00-00-00-00-00-00'
    option interface_id '1'
    option load_balance_num '4'
    option vrf_id '0'
    option port_id '0'
    option action 'rdtcpu'
    option mirror 'no'
    option counter 'no'

config switch_ext
    option device 'switch0'
    option name 'IpHostentry'
    option entry_id '0'
    option entry_flags '2'
    option entry_status '7'
    option ip_addr '2::9'
    option mac_addr '00-00-00-00-00-00'
    option interface_id '1'
    option load_balance_num '4'
    option vrf_id '0'
    option port_id '0'
    option action 'rdtcpu'
    option mirror 'no'
    option counter 'no'

config switch_ext
    option device 'switch0'
    option name 'IpHostentry'
    option entry_id '0'
    option entry_flags '2'
    option entry_status '7'
    option ip_addr '3::9'
    option mac_addr '00-00-00-00-00-00'
    option interface_id '1'
```

```
option load_balance_num '4'
option vrf_id '0'
option port_id '0'
option action 'rdtcpu'
option mirror 'no'
option counter 'no'

config switch_ext
    option device 'switch0'
    option name 'IpHostentry'
    option entry_id '0'
    option entry_flags '2'
    option entry_status '7'
    option ip_addr '4::9'
    option mac_addr '00-00-00-00-00-00'
    option interface_id '1'
    option load_balance_num '4'
    option vrf_id '0'
    option port_id '0'
    option action 'rdtcpu'
    option mirror 'no'
    option counter 'no'

config switch_ext 'acl'
    option device 'switch0'
    option name 'AclUdfprofile'
    option port '1'
    option user_defined_type '13'
    option user_defined_offset '22'
    option user_defined_length '2'

config switch_ext 'acl'
    option device 'switch0'
    option name 'AclUdfprofile'
    option port '1'
    option user_defined_type 'l3plus'
    option user_defined_offset '31'
    option user_defined_length '13'

config switch_ext
    option device 'switch0'
    option name 'AclRule'
    option rule_id '10'
    option priority '10'
```

```

option rule_type 'ip4'
option ip_protocol '0x2f'
option ip_protocol_mask '0xff'
option user_defined_field_value '00-44'
option user_defined_field_mask 'ff-ff'
option action_arp_ptr '0x40'
option port_bitmap '0x2'

config switch_ext
    option device 'switch0'
    option name 'AclRule'
    option rule_id '20'
    option priority '20'
    option rule_type 'ip6'
    option ip_protocol '0x2f'
    option ip_protocol_mask '0xff'
    option user_defined_field_value '00-00-00-00-00-00-00-00-00-00-
00-00-00-44'
    option user_defined_field_mask '00-00-00-00-00-00-00-00-00-00-
00-00-ff-ff'
    option action_arp_ptr '0x21'
    option port_bitmap '0x2'

```

## 6.27 Multiple PPPoE sessions

This section describes the requirements and software design for supporting multiple PPPoE sessions and dual WAN PPPoE.

In NSS, eth0 is a WAN Ethernet port. To add a second WAN port, we can take up any free Ethernet port and add configuration into /etc/network/config file. The details of the configuration are in the design section.

Multiple PPPoE sessions are established when there are multiple PPPoE servers running on the WAN side. The PPPoE server is created on a separate machine connected to the WAN port. The LAN port of the IPQ8064 (DUT) connects clients to PPPoE clients running on the IPQ8064 board.

The PPPoE client session runs on the IPQ8064/DUT board. The traffic is accelerated from LAN to WAN and WAN to LAN through these PPPoE tunnels/sessions. The PPPoE server also plays an important role here since it has also to route the packet to the right destination machine.

The following are the requirements for Dual WAN ports:

- Both the WAN port must support PPPoE sessions with fast path (NSS acceleration) and slow-path or one fast path and one slow path.
- Both the WAN ports must support same set of acceleration as supported with single WAN port and single PPPoE session.

The following are the requirements for Multiple PPPoE sessions:

- Accelerate up to four PPPoE sessions through NSS.
- PPPoE sessions must work with ECM (Enhanced Connection Manager).
- Single PPPoE features need to be supported with multiple PPPoE.
- IPv6 and IPv4 must support the PPPoE multiple sessions.

NSS already supports one PPPoE sessions over a single WAN port. To extend the support to add up to four PPPoE sessions and dual WAN port, we will need to add configuration changes in /etc/config/network file. NSS FW supports up to eight PPPoE sessions in the NSS FW with verification performed for four PPPoE sessions.

First, a PPPoE server must be set up. It is recommended to set up a single PPPoE server over a physical interface. Note that physical interface must not be configured and it needs to be in UP state. The PPPoE session is established over the physical interface.

### 6.27.1 Start a PPPoE server

The ipaddress\_pool file will specific the range of IP address that the server can give to clients.

The /etc/ppp/chap-secrets file will specify the client,server,secret, and IP addresses to be used to authenticate client.

```
#!/bin/bash
#####
# Simple script that starts PPPoE Server
#####

# Enable IP Forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

# Start PPPoE Server
pppoe-server -C isp -L 192.168.3.1 -p /etc/ppp/ipaddress_pool -I eth1 -m 1412

# Set Firewall rules
iptables -t nat -F POSTROUTING
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

For creating more than one PPPoE server, please change the physical interface through -I option and change the IP addresses too using -L option and a different ipaddress\_pool file.

### 6.27.2 Stop a PPPoE server

```
#!/bin/bash

#####
# Simple script that stops PPPoE Server
#####

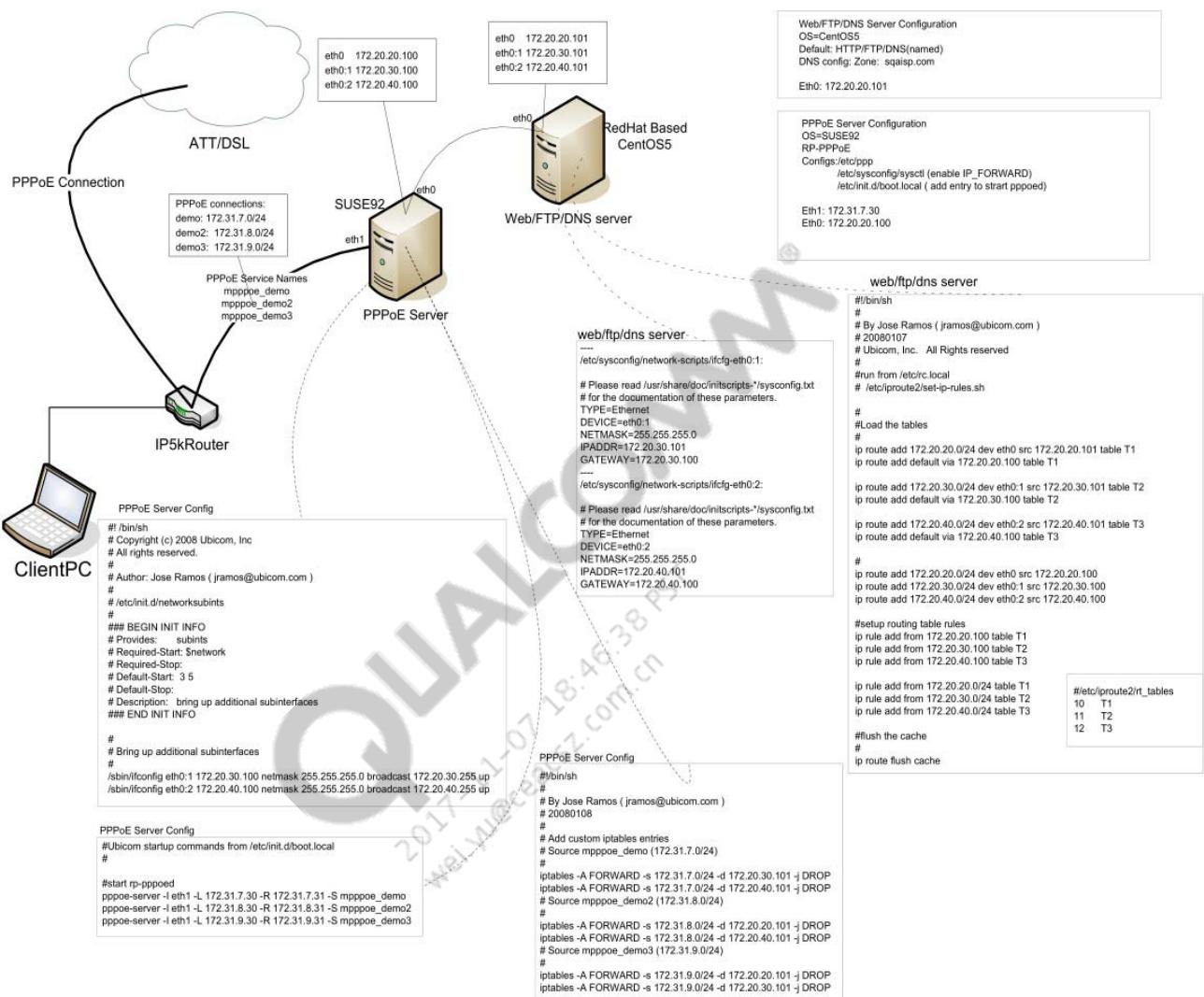
# Disable IP Forwarding
echo 0 > /proc/sys/net/ipv4/ip_forward

# Kill PPPoE Server
killall pppoe-server
killall pppd

# Flush the IPtable rules.
iptables -t nat -F POSTROUTING
```

### 6.27.3 Sample configuration

The IP5kRouter in the following diagram indicates a DUT or IPQ8064 board:



## 6.27.4 Create multiple PPPoE sessions

To create four PPPoE sessions, add the following lines in /etc/config/network file of the DUT:

```

config interface 'wan0'
    option ifname 'eth0'
    option proto 'pppoe'
    option username builder1
    option password 12341

config interface 'wan1'
    option ifname 'eth0'
    option proto 'pppoe'
    option username builder2
    option password 12342
  
```

```

        config interface 'wan2'
            option ifname 'eth0'
            option proto 'pppoe'
            option username builder3
            option password 12343

        config interface 'wan3'
            option ifname 'eth0'
            option proto 'pppoe'
            option username builder4
            option password 12344
    
```

Make sure to restart network service as follows:

```
# /etc/init.d/network restart
```

### 6.27.5 Create Dual WAN port

To create dual WAN ports, all you need is to specify the second physical Ethernet port in the /etc/config/network file.

```

        config interface 'wan0'
            option ifname 'eth0'
            option proto 'pppoe'
            option username builder
            option password 1234

        config interface 'wan1'
            option ifname 'eth2'
            option proto 'pppoe'
            option username builder
            option password 1234
    
```

Make sure to restart network service as follows:

```
# /etc/init.d/network restart
```

### 6.27.6 Display PPPoE statistics

Obtain the statistics by running following commands:

```
# ifconfig pppo-wanX (X is the number specified in the /etc/config/network file)
```

Also, sar -n DEV 2 | egrep "IFACE|ppp" can be run to see the throughput and PPS on these PPPoE clients.

NSS host also provides these statistics however they are maintained by NSS FW:

```
# cat /sys/kernel/debug/qca-nss-drv/stats/pppoe
```

### 6.27.7 UCI commands

```
uci set network.loopback=interface  
uci set network.loopback.ifname=lo  
uci set network.loopback.proto=static  
uci set network.loopback.ipaddr=127.0.0.1  
uci set network.loopback.netmask=255.0.0.0  
uci set network.lan=interface  
uci set network.lan.ifname='eth1'  
uci set network.lan.type=bridge  
uci set network.lan.proto=static  
uci set network.lan.ipaddr=192.168.1.1  
uci set network.lan.netmask=255.255.255.0  
uci set network.lan.ip6addr=2aaa::1/64  
uci set network.wan0=interface  
uci set network.wan0.ifname=eth0  
uci set network.wan0.proto=pppoe  
uci set network.wan0.username='builder'  
uci set network.wan0.password='builder'  
uci set network.wan1=interface  
uci set network.wan1.ifname=eth0  
uci set network.wan1.proto=pppoe  
uci set network.wan1.username='builder1'  
uci set network.wan1.password='builder1'  
uci set network.wan2=interface  
uci set network.wan2.ifname=eth0  
uci set network.wan2.proto=pppoe  
uci set network.wan2.username='builder2'  
uci set network.wan2.password='builder2'  
uci set network.wan3=interface  
uci set network.wan3.ifname=eth0  
uci set network.wan3.proto=pppoe  
uci set network.wan3.username='builder3'
```

```
uci set network.wan3.password='builder3'
uci set firewall.@defaults[0]=defaults
uci set firewall.@defaults[0].syn_flood=1
uci set firewall.@defaults[0].input=ACCEPT
uci set firewall.@defaults[0].output=ACCEPT
uci set firewall.@defaults[0].forward=ACCEPT
uci set firewall.@zone[0]=zone
uci set firewall.@zone[0].name=lan
uci set firewall.@zone[0].network=lan
uci set firewall.@zone[0].input=ACCEPT
uci set firewall.@zone[0].output=ACCEPT
uci set firewall.@zone[0].forward=ACCEPT
uci set firewall.@zone[1]=zone
uci set firewall.@zone[1].name=wan
uci set firewall.@zone[1].network=wan
uci set firewall.@zone[1].input=ACCEPT
uci set firewall.@zone[1].output=ACCEPT
uci set firewall.@zone[1].forward=ACCEPT
uci set firewall.@zone[1].masq=1
uci set firewall.@zone[1].mtu_fix=1
uci add firewall redirect
uci set firewall.@redirect[0]=redirect
uci set firewall.@redirect[0].target=DNAT
uci set firewall.@redirect[0].src=wan
uci set firewall.@redirect[0].dest=lan
uci set firewall.@redirect[0].proto='tcp udp'
uci set firewall.@redirect[0].name=DNAT
uci set firewall.@redirect[0].src_ip=192.168.2.2
uci set firewall.@redirect[0].src_dip=172.31.7.2
uci set firewall.@redirect[0].dest_ip=192.168.1.2
uci add firewall redirect
uci set firewall.@redirect[1]=redirect
```

```
uci set firewall.@redirect[1].target=DNAT
uci set firewall.@redirect[1].src=wan
uci set firewall.@redirect[1].dest=lan
uci set firewall.@redirect[1].proto='tcp udp'
uci set firewall.@redirect[1].name=DNAT
uci set firewall.@redirect[1].src_ip=192.168.2.3
uci set firewall.@redirect[1].src_dip=172.31.7.3
uci set firewall.@redirect[1].dest_ip=192.168.1.3
uci commit
uci add firewall redirect
uci set firewall.@redirect[2]=redirect
uci set firewall.@redirect[2].target=DNAT
uci set firewall.@redirect[2].src=wan
uci set firewall.@redirect[2].dest=lan
uci set firewall.@redirect[2].proto='tcp udp'
uci set firewall.@redirect[2].name=DNAT
uci set firewall.@redirect[2].src_ip=192.168.2.4
uci set firewall.@redirect[2].src_dip=172.31.7.4
uci set firewall.@redirect[2].dest_ip=192.168.1.4
uci commit
uci add firewall redirect
uci set firewall.@redirect[3]=redirect
uci set firewall.@redirect[3].target=DNAT
uci set firewall.@redirect[3].src=wan
uci set firewall.@redirect[3].dest=lan
uci set firewall.@redirect[3].proto='tcp udp'
uci set firewall.@redirect[3].name=DNAT
uci set firewall.@redirect[3].src_ip=192.168.2.5
uci set firewall.@redirect[3].src_dip=172.31.7.5
uci set firewall.@redirect[3].dest_ip=192.168.1.5
uci commit
```

Make sure to restart both firewall and network service:

```
/etc/init.d/firewall restart
```

```
/etc/init.d/network restart
```

If you need to enable debugging, then uncomment debug line in /etc/ppp/options file:

```
echo '#debug' > /etc/ppp/options
echo 'logfile /dev/null' >>/etc/ppp/options
echo 'noipdefault'>>/etc/ppp/options
echo 'noaccomp'>>/etc/ppp/options
echo 'nopcomp'>>/etc/ppp/options
echo 'nocrtscts'>>/etc/ppp/options
echo 'lock'>>/etc/ppp/options
echo 'maxfail 0'>>/etc/ppp/options
echo 'lcp-echo-failure 10000'>>/etc/ppp/options
echo 'lcp-echo-interval 1'>>/etc/ppp/
```

## 6.27.8 Add routes

Adding route is the most important thing. Make sure you don't have conflicting routes otherwise packets will be dropped in the DUT and wouldn't reach PPPoE server. Similarly, PPPoE server also needs to have route so that it can forward the packet to right machine.

```
# ip route add 192.168.2.2 dev pppoe-wan0
#ip route add 172.31.7.1 dev pppoe-wan0
```

```
#ip route add 192.168.2.3 dev pppoe-wan1
# ip route add 172.31.8.1 dev pppoe-wan1
```

```
#ip route add 192.168.2.4 dev pppoe-wan2
#ip route add 172.31.9.1 dev pppoe-wan2
```

```
#ip route add 192.168.2.5 dev pppoe-wan3
#ip route add 172.31.10.1 dev pppoe-wan3
```

Advanced routing or policy based routing can be accelerated through NSS in a dual-WAN configuration. In particular, the following command is supported in a dual wan configuration in AP161:

```
ip route replace default nexthop via 192.xxx.xxx.xx0 dev eth0 weight 1 nexthop via 192.xxx.xxx.xx1  
dev eth1 weight 1
```

This config script provides load-balancing for dual WAN interfaces.

QUALCOMM®  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

# 7 Rate Control Features

---

This chapter describes the rate control functionalities and techniques, such as 256 QAM rate support in 2.4 GHz operation, coordinated ATF between RootAP & Repeater, disable selected legacy rates, and content aware routing.

## 7.1 Rate Control

Rate Control is the algorithm used by the WLAN software to choose the rate for transmitting a frame to a STA.

For a direct attach data path, software must select a rate series, which consists of four rates and the number of attempts (retries) at each rate. The rate series is used by the WLAN MAC hardware. The MAC hardware first attempts to transmit at the rate of the first entry in the series. If it fails, the transmission is repeated at that rate for a total number of attempts equal to the number of retries specified in the series. If all attempts at that rate fail, the MAC hardware drops down to the next rate in the series and repeats. If all attempts at all rates fail, the MAC hardware informs software by marking the status of the frame transmission as excessive retries.

Rate Control estimates the best rate to use for transmission based on statistics collected from previous transmissions to that STA. It uses the best rate as the first (highest) rate of the rate series. This first and highest rate dictates the other rates and the retries in the rate series.

For an offload data path, software must select a rate series, which consists of three rates for 20 MHz, 40 MHz and 80 MHz, and the number of attempts (retries). The rate series is used by the WLAN MAC hardware. The MAC hardware based on whether the extension channel CCA decides to use a 20 MHz, 40 MHz or 80 MHz rate. In another optional configuration, hereafter referred to as static bandwidth mode, the software rate control module only programs one bandwidth rate. After a specified number of hardware retries, (typically very low, such as 1) the MAC hardware indicates a failure to software, which passes statistics to the rate control module, which can potentially lower the rate. Subsequent retries are performed with a lower rate. The retries are therefore handled in software.

Rate Control estimates the best rate to use for transmission based on statistics collected from previous transmissions to that STA. In offload design, a rate control module also manages dynamic bandwidth selection, which allows a particular bandwidth to be used for a given STA, for example, when an extension channel becomes busy on the receive side, the transmitter based on higher frame error rate may drop to a lower bandwidth.

### 7.1.1 Theory of Operation

The objective of Rate Control is to determine the transmission rate to a STA that maximizes the MAC layer throughput to that STA. In offload design, the rate control module implements

dynamic bandwidth selection, and a given bandwidth selection of best rate based on application goodput under given channel conditions.

The MAC Layer Throughput is given by the equation:

$$\text{MAC Layer Throughput} = \text{PHY Rate} * (1 - \text{PER}) \text{ where}$$

- PHY Rate is the physical layer rate (for example, MCS0–MCS9 for VHT, MCS0–MCS23 for 802.11n, or Legacy OFDM 6 Mb/s to 54 Mb/s for 802.11a/g, or CCK 1Mb/s to 11 Mb/s for 802.11b, or a combination of these rates for 802.11g/b/n)
- PER (Packet Error Rate) is the fraction of transmitted packets that are lost at the link layer.

For example, at rate MCS15, if a packet is sent 100 times and 10 frames fail, the PER for MCS15 =  $10/100 = 0.1$ . It is expressed as a fraction or percentage. In practice, a frame is sent only a few times, so the rate control module needs to estimate PER.

For a STA at a given location, PER is usually an increasing function of the rate (if link budget is the only source of packet loss; that is, neglecting collisions or interference loss). Therefore, the MAC Layer throughput is a concave function of rate. The rate control algorithm attempts to determine the PHY Rate that maximizes the MAC layer throughput.

Rate control operates by collecting statistics about previous transmissions to a STA. It uses these statistics to estimate the best rate and bandwidth to use. In addition, it uses two other mechanisms to influence which rate and bandwidth are chosen: probing at higher rates, and bandwidth and aging per STA statistics.

The following are the two main operations that occur as part of the Rate Control processing:

- *RateFind* is the sequence of operations used to determine the rate and bandwidth to use for transmission to a STA. It is executed before the transmission of a frame; that is, before software hands the frame to the MAC hardware for transmission.
- *RateUpdate* is the sequence of operations used to store statistics from the result of a frame transmission to a STA. It is executed after transmission of a frame; that is, when the MAC hardware informs software that transmission is complete.

The implementation section describes these two operations and the data structures needed to support them.

## 7.1.2 Direct Attach Implementation

### 7.1.2.1 Rate Control Data

Rate control maintains several data structures to assist in rate selection. The main data structures are:

1. Rate table (struct RATE\_TABLE):

The Rate Table is a table that stores information about the PHY Rates for a given mode (802.11na, 802.11ng, 802.11a, 802.11g, 802.11b). The Rate Table contains information for all rates for a given mode (802.11na, 802.11ng, and so on). This information includes the rate code, the actual rate in Mb/s, whether the rate is valid for UAPSD operation, valid for STBC, and so forth. A single instance of this table is created for a VAP.

The WLAN software uses a value called the *rate code* specified by the hardware to select the rate to be used for transmission of a particular frame. It fills the rate code in the rate field in the hardware descriptor for the frame. The value ranges from 0 to a number corresponding to the maximum PHY rate (combination of modulation/coding scheme).

An 802.11 PHY Rate is a combination of Modulation and Coding Scheme (for example, MCS0-MCS15). However, in addition to the MCS, there are other parameters which control the actual physical rate such as channel width (HT40 or HT20) and Guard Interval [GI] (half or full). The Rate Table contains entries for every such combination. The base index specifies the modulation and coding rate. The index in the table is called the *base index*. The *rate index* is another entry in the rate table that has the same modulation and coding rate, but may have a different channel width and/or guard interval.

The Rate Control software uses the Rate Control Table for two purposes:

- When transmitting a frame, the Rate Control Algorithm must determine if a given rate can be used to transmit, subject to constraints on specific parameters such as channel width, or whether STBC is supported. To do this, it checks every entry in the rate table to see if it is the best rate to use based on the STA PER and if it is valid based on the constraints.
  - When updating the per STA Rate Control Information (see below). When a transmission completes, the LMAC finds the base index for the rate which was used for the transmission from its data structures. The Rate Control software uses the same number (called the *baseIndex*) to select a base entry in the Rate Table. It then uses a set of indices in the base entry to find the rate index. The indices (*cwmIndex*, *sgiIndex*) are used to find the rate index for the particular rate. The index approach keeps the lookup time bounded by the order of the number of parameters (CW, GI), rather than the number of entries in the rate table. Thus the input to the lookup function is the base index along with the channel flags. The output is the rate index.
2. Per STA Rate Control Information and Statistics (struct *atheros\_node*): This structure contains information about a particular STA. There are as many instances of this structure as STAs associated with the AP. The fields include:
    - STA Information: Valid (supported) rates, pointers to LMAC information
    - STA Statistics: (struct *TxRateCtrlState*): The structure contains information for a particular STA. It includes PER estimate for every rate, maxPHY Rate for the STA, probeRate, probe Interval, lastProbeTime

### 7.1.2.2 Rate Control Functions

#### **rcFind\_ht()**

The RateFind() function reads each entry in the array of valid PHY rates for a STA. For each rate, it calculates the MAC layer throughput based on the current PER estimate of the rate from the per STA Rate Control statistics. The PER is lower bounded by a rate of 12% to account for collisions in a TCP connection between TCP ACKs and data packets.

The function chooses the rate that maximizes the MAC layer throughput, subject to the constraint that the rate must be lower than the MaxPhyRate of the STA.

The STA maxPhyRate (stored in the per STA Rate Control Statistics) for a STA is an estimate of the maximum PHY rate that can be used to transmit to the STA. It is estimated based on the AP statistics of transmissions to that STA. It is a dynamically changing variable maintained in the per STA rate control statistics structure based on statistics. The maxPhyRate is not necessarily the same as the rate which is eventually chosen for transmission, because rate control tries to maximize the MAC layer throughput rather than the physical rate. It serves as a bound on the search to ensure that RateControl does not choose a rate that is too high without first probing it. The maxPhyRate is a protection mechanism against the limitations of the PER estimation, aging, and monotonicity correction (see [Section](#) on RateUpdate).

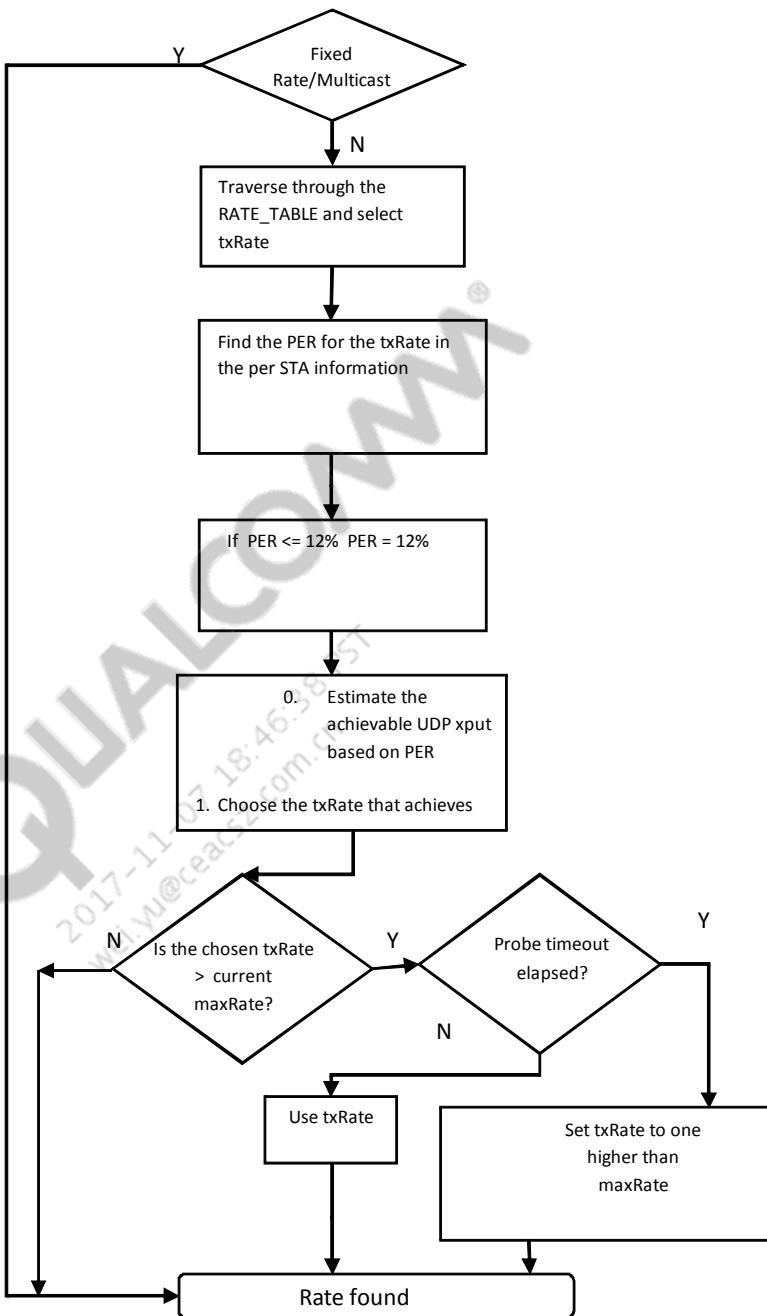
Rate Control periodically “probes” higher rates to see if the channel conditions have improved. It does this by selecting a rate higher than the chosen rate for a normal frame begin sent to the STA. A probe is sent if the selected rate is equal to the maxPhyRate allowed, and if the probe Interval has elapsed.

At startup the maxPhyRate is set to a rate close to but not equal to the maximum valid PHY rate and the PER for all rates is set to 0. This means that the first rate chosen will be the maxPhyRate. Eventually, as the RateControl module gathers statistics about the transmissions, the algorithm will converge to the rate which maximizes MAC Layer Throughput.

If the frame is a broadcast or multicast frame, a fixed rate (equal to the lowest rate) is used and the algorithm is not exercised.

### **rcFind\_ht() Algorithm**

- Search through the per STA Rate Control Statistics (from lowest PHY Rate to the maxValidRate for that STA)
  - If the Rate < maxPhyRate
    - Select PHY Rate and lookup the corresponding PER.  
Lower bound the PER (the lower bound is 12% PER because of collisions between TCP data and ACKs).
    - Find the PHY Rate that maximizes MAC Layer Throughput. This is the chosen rate.
    - If PER is excessive (>55%), chose the next lower rate as the chosen rate.
- Check if the frame should be used as a probe:
  - If the rate chosen is the maxPHYRate, AND its PER is less than the minimum bound (12% TCP PER), AND the probe interval has expired: set the rate to a rate one higher than the maxPhyRate
  - If the frame is a probe frame, only the first attempt (that is, the first entry of the rate series) will be set to the probe rate.

**Figure 7-1 rc\_Find\_ht() Algorithm****rcUpdate\_ht()**

The RateUpdate() function is called once for every rate in the rate series that has been attempted. The PER is updated differently depending on whether it is an intermediate rate or the final rate. An Excessive Retry is defined as a transmission attempt where all rates in the rate series have failed.

for all retries. An Intermediate Failure is where all attempts at one rate in the rate series have failed, but a lower rate in the rate series has succeeded.

The RateUpdate() function updates statistics on completion of transmission of a frame.

The PER is calculated for the rate at which transmission was attempted. The calculation of the PER differs depending on whether the first transmission attempt succeeded, an intermediate transmission succeeded, whether there were excessive retries (all attempts failed), or if the frame was a probe frame. In addition, a Low Pass Filter is applied to the new estimate to smooth out the transients and take into account the past history.

In addition, this function also modifies the statistics for the entire per STA statistics in the following ways:

- Monotonicity of PER: The function maintains the PER to be as a monotonically increasing function of rate. This follows the intuitive reasoning that if the channel improves for one rate, it must have improved correspondingly for all rates. If the monotonic increasing property is not maintained, then the Rate Control algorithm will chose the subsequent rates incorrectly.
- Aging of PER: The PER is aged out gradually; that is, at a specific time interval, the PER for all rates is reduced.
- maxPhyRate: If the PER for the rate used for transmission is abnormally high (>55%), then the maxPhyRate is dropped to a lower chosen. If the maxPhyRate has worse MAC Layer Throughput than the next lower rate, the maxPhyRate is reduced. If the frame was a probe and succeeded then the maxPhyRate is increased.

### **RCUpdate() Algorithm**

- If the rate has failed because of RTS failures, do not update the Rate, return
  - RTS failure may indicate failure for reasons other than Rate Control
- Update PER statistics
  - Calculate the instantaneous PER for the rate based on the number of retries from a lookup table
  - If there has been Excessive Retries for the rate
    - PER = old PER + 0.3, upper bounded by 1
  - If it is an Intermediate Failure but not Excessive Retries
    - PER =  $7/8 * \text{old PER} + (1/8) * 1$
  - If it is neither an Intermediate Failure nor Excessive Retries
    - If some failures within the rate
 
$$\text{PER} = 7/8 * \text{old PER} + 1/8 * \text{instantaneous PER}$$

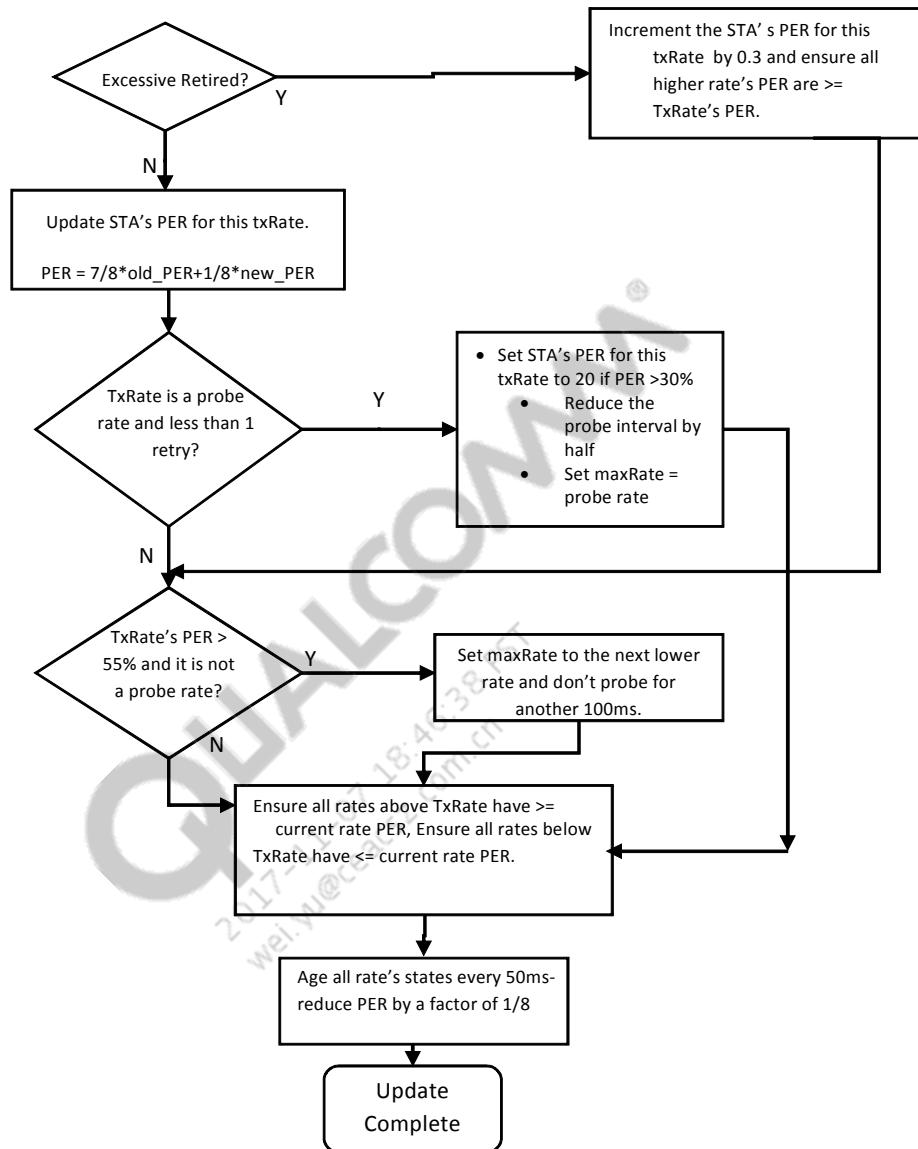
Use failures to determine the instantaneous PER

      - If no failure within the rate
 
$$\text{PER} = 7/8 * \text{old PER} + 1/8 * \text{instantaneous PER}$$

Use lookup table and number of retries at previous rate to determine instantaneous PER

- Update maxPhyRate statistics
  - If the frame was a probe frame and the number of retries  $\leq 1$ , set maxPhyRate to the probe rate
  - If the PER for the chosen rate is excessive ( $> 55\%$ ), then reduce the maxPhyRate to the next lower rate
  - Based on the current PER estimates, check if the maxPhyRate has worse MAC Layer Throughput than the next lower rate
  - If yes, reduce the maxPhyRate to the next lower rate
- Check for monotonically increasing property of per STA statistics
  - Keep the PER in the per STA PER estimate table monotonically increasing with rate. For rates  $<$  current rate, the PER must be less than or equal to the current rates' PER. For rates  $>$  current rate, the PER must be greater than or equal to the current rates' PER.
- Age out the per STA statistics
  - Age the PER: At specific intervals, reduce the PER:  $\text{PER} = 7/8 * \text{PER}$

QUALCOMM  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn



**Figure 7-2 RCUpdate() Algorithm**

### 7.1.3 Offload Implementation

#### 7.1.3.1 Rate Control Data

Rate control maintains several data structures to assist in rate selection. The main data structures are:

1. Hardware rate table (struct WHAL\_RATE\_TABLE):

The Rate Table is a table that stores information about the PHY rates for all the supported modes or PHYs (11ac, 11n, 11a, 11bg). This rate table, in addition to all the standard information such as bits per symbol rates in kbps, short GI, LGPC, and STBC, stores hardware

rate code expected by the MAC hardware. The fields in this table are used as lookup tables to derive rate related information by the rate control.

The WLAN software uses a value called the *rate code* specified by the hardware to select the rate to be used for transmission of a particular frame. It fills the rate code in the rate field in the hardware descriptor for the frame. The value ranges from 0 to a number corresponding to the maximum PHY rate (combination of modulation/coding scheme).

An 802.11 PHY Rate is a combination of Modulation and Coding Scheme (for example, MCS0-MCS23 for HT and MCS0...MCS9 for VHT). However, in addition to the MCS, there are other parameters which control the actual physical rate such as channel width (HT40 or HT20, VHT80, VHT40, VHT20) and Guard Interval [GI] (half or full). The Rate Table contains entries for every such combination. The base index specifies the modulation and coding rate. The index in the table is called the *base index*. The *rate index* is another entry in the rate table that has the same modulation and coding rate, but may have a different channel width and/or guard interval.

The Rate Control software uses the Rate Control Table for the following:

- When transmitting a frame, the Rate Control Algorithm must determine if a given rate can be used to transmit, subject to constraints on specific parameters such as channel width, or whether STBC is supported. To do this, it checks every entry in the rate table to see if it is the best rate to use based on the STA PER and if it is valid based on the constraints.
- When updating the per STA Rate Control Information (see item 2 below). When a transmission completes, the LMAC finds the base index for the rate which was used for the transmission from its data structures. The Rate Control software uses the same number (called the *baseIndex*) to select a base entry in the Rate Table. It then uses a set of indices in the base entry to find the rate index. The indices (*cwmIndex*, *sgiIndex*) are used to find the rate index for the particular rate. The index approach keeps the lookup time bounded by the order of the number of parameters (CW, GI), rather than the number of entries in the rate table. Thus the input to the lookup function is the base index along with the channel flags. The output is the rate index.
- Get the maximum number of bytes that can be packed in AMPDU for a given rate.

## 2. Per Vdev Rate Control Information (struct RATE\_CONTEXT):

This stores information that are specific to vdev. For AP Vdev this stored properties such as what rates (MCS), bandwidth, SGI, LDPC are supported by the AP vdev or not. Each associated STA in the AP Vdev may have different properties that are stored in per peer rate control information as indicated in item 3 below.

## 3. Per STA Rate Control Information and Statistics (struct rate\_node): This structure contains information about a particular STA. There are as many instances of this structure as STAs associated with the AP. The fields include:

- STA Information: Valid (supported) rates, pointers to LMAC information
- STA Statistics: (struct TxRateCtrlState): The structure contains information for a particular STA. It includes:
  - PER estimate for every rate, maxPHY Rate for the STA, probeRate, probe Interval, lastProbeTime

### 7.1.3.2 Rate Control Functions

#### RATE\_GetTxRetrySched

The RATE\_GetTxRetrySched() function returns the rate schedule to be used by the WLAN software to program rate-related information in the transmit descriptor of the PPDU given to the MAC hardware for the transmission. For each rate, it calculates the MAC layer throughput based on the current PER estimate of the rate from the per STA Rate Control statistics.

The function chooses the rate, bandwidth, and number of spatial streams that maximize the MAC layer throughput, subject to the constraint that the rate must be lower than the MaxPhyRate of the STA, for the selected spatial stream configuration.

The STA maxPhyRate (stored in the per STA Rate Control Statistics) for a STA is an estimate of the maximum PHY rate that can be used to transmit to the STA for a given spatial stream configuration. It is estimated based on the AP statistics of transmissions to that STA. It is a dynamically changing variable maintained in the per STA rate control statistics structure based on statistics. The maxPhyRate is not necessarily the same as the rate which is eventually chosen for transmission, because rate control tries to maximize the MAC layer throughput rather than the physical rate. It serves as a bound on the search to ensure that RateControl does not choose a rate that is too high without first probing it. The maxPhyRate is a protection mechanism against the limitations of the PER estimation, and monotonicity correction (see the *rcUpdate\_ht()* subsection).

Rate Control periodically “probes” higher rates to see if the channel conditions have improved. It does this by selecting a rate higher than the chosen rate for a normal frame being sent to the STA. A probe is sent if the probe interval has elapsed. Probes are also periodically sent using a different numbers of spatial streams to determine, if another spatial stream configuration would give better throughput. The modulation rate selected for alternate spatial stream probes is the modulation rate that can potentially achieve throughput better than the current modulation rate for the current spatial stream configuration. In general, this modulation rate will not be the same modulation rate used for the current spatial stream configuration.

On peer association, the maxPhyRate is set to the minimum valid PHY rate. The PER for all rates is set to 0. Eventually, as the RateControl module gathers statistics about the transmissions, the algorithm will converge to the rate and spatial stream configuration which maximizes MAC layer throughput. Starting with a lower rate helps avoid any frame loss when higher rates do not work.

If the frame is a management frame, broadcast or multicast frame, a fixed rate (equal to the lowest rate) is used and the algorithm is not exercised.

## Algorithm

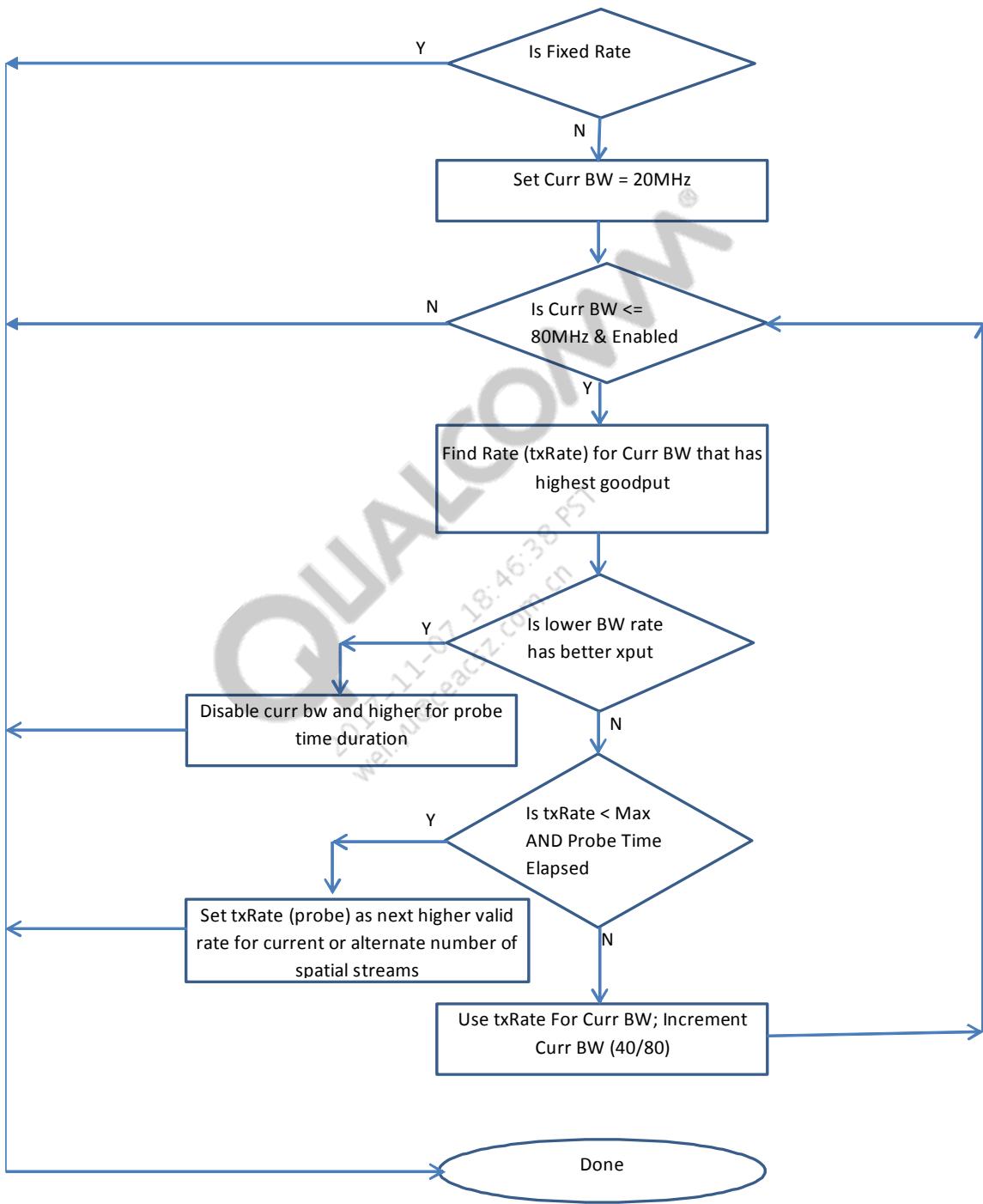
- Search through the per STA Rate Control Statistics (from lowest PHY Rate to the maxValidRate for that STA)
  - Find the PHY Rate and number of spatial streams that maximizes MAC Layer Throughput. This is the chosen rate. Note that hysteresis is applied and a new spatial stream configuration will not be selected unless the potential MAC throughput gain compared against the previous spatial stream configuration would be greater than 10%.
  - If the throughput achieved on the current bandwidth is less than the throughput achieved on a lower bandwidth, do not use the higher bandwidth for a while, or
  - If bandwidth is higher than 20MHz, running at lowest MCS (MCS0), and PER > 30%, do not use this and higher bandwidths for a while.
- Probing higher rates

If the current rate is less than the maximum supported rate, 4 PPDUs have been successfully transmitted on the current rate, and a probe period has elapsed since the last probe was sent, select a higher valid rate with the current spatial stream configuration to probe. The aggregate size is reduced to 4 subframes. Using aggregate instead of singles gives more statistics to decide whether the probe has succeeded or failed. Probing is also performed with more or fewer spatial streams, where the modulation rate selected is based on the throughput of the currently selected spatial stream configuration.

- Probing higher bandwidths

When a higher bandwidth is disabled, a probe is pending, the bandwidth probe interval has expired and is currently not running on lowest MCS, but has successfully transmitted 4 frames, find a probe rate on a higher bandwidth that has more throughput than the rate on the current lower bandwidth MCS.

If the frame is a probe frame, only one hardware try will be set.

**Figure 7-3 RATE\_GetTxRetrySched Algorithm**

### **rcUpdate\_ht()**

The RateUpdate() function is called once for every rate in the rate series that has been attempted. The PER is updated differently depending on whether it is an intermediate rate or the final rate. An Excessive Retry is defined as a transmission attempt where all rates in the rate series have failed for all retries. An Intermediate Failure is where all attempts at one rate in the rate series have failed, but a lower rate in the rate series has succeeded. The RateUpdate() function updates statistics on the completion of transmission of a frame.

The PER is calculated for the rate at which transmission was attempted. The calculation of the PER differs depending on whether the first transmission attempt succeeded, an intermediate transmission succeeded, whether there were excessive retries (all attempts failed), or if the frame was a probe frame. In addition, a Low Pass Filter is applied to the new estimate to smooth out the transients and take into account the past history.

In addition, this function also modifies the statistics for the entire per STA statistics in the following ways:

- Monotonicity of PER: The function maintains the PER to be as a monotonically increasing function of rate and number of spatial streams. This follows the intuitive reasoning that if the channel improves for one rate, it must have improved correspondingly for all rates. If the monotonic increasing property is not maintained, then the Rate Control algorithm will choose the subsequent rates incorrectly.
- Monotonicity of maxPhyRate: The function maintains the maxPhyRate for each spatial stream configuration to be monotonically decreasing as the number of spatial streams increases.
- maxPhyRate: If the PER for the rate used for transmission is abnormally high (>55%), then the maxPhyRate is dropped to a lower rate. If the frame was a probe and it succeeded, then the maxPhyRate is increased.

### **RCUpdate() Algorithm**

- If the rate has failed because of RTS failures, do not update the rate, return
  - RTS failure may indicate failure for reasons other than Rate Control
- Update PER statistics
  - Calculate the instantaneous PER for the rate based number of failures in the aggregate transmission
  - Update PER for the remote station
  - Update maxPhyRate statistics
    - If the frame was a probe frame and the number of retries  $\leq 1$ , set maxPhyRate to the probe rate
    - If the PER for the chosen rate is excessive (> 55%), then reduce the maxPhyRate to the next lower rate
  - Check for monotonically increasing property of per STA statistics
    - Keep the PER in the per STA PER estimate table monotonically increasing with rate. For (modulation rate AND number of spatial streams) < current rate, the PER must be less than or equal to the current rates' PER. For (modulation rate AND number of

spatial streams) > current rate, the PER must be greater than or equal to the current rates' PER.

- Keep the maxPhyRate monotonically decreasing with number of spatial streams.  
For example, the maxPhyRate for a 2 spatial stream configuration must be less than or equal to the maxPhyRate for the 3 spatial stream configuration.

## 7.1.4 Code

### 7.1.4.1 Direct Attach

The code for rate control is in the folder: drivers/wlan/lmac/ratectrl.

The files for rate control are:

- ratectrl\_11n.c: This file contains the rateFind() and rateUpdate() functions
- ar5416Phy.c: This file contains the rate table for AR5416 and later chips (all 11n)
- ar5212Phy.c: This file contains the rate table for AR5212 and 11g only chips
- ratectrl.h: Contains definitions of data structures.
- if\_athrate.c: This file contains the functions which the rate control modules exports as an API for the driver LMAC to use.

### 7.1.4.2 Offload Code

The code for Rate Control is in the folder: drivers/wlan/lmac/ratectrl. The files for Rate Control are:

- ratectrl\_11ac.c: This file contains the RATE\_GetTxRetrySched() and rateUpdate() functions
- ar600P\_Phys.c: This file contains the rate table for AR9888 and later chips (all 11ac)
- ratectrl.h: Contains definitions of data structures.
- ar\_wal\_rc.c contains APIs that are primarily used when running with fixed rate and rate selection for group-addressed data and management, control frames.

## 7.1.5 API

The file if\_athrate.c contains functions which define the interface provided by the Rate Control module to the external world. These functions are called from the LMAC transmit path (ath\_xmit.c/ath\_edma\_xmit.c). The API consists of the following main functions:

- ath\_rate\_findrate(): This function eventually calls the rcRateFind\_ht() which returns the rate series to use for transmission to a STA based on current state.
- ath\_rate\_tx\_complete(): This function eventually calls either the rxRateUpdate\_ht() [for 11n, or legacy rate update function for 11g] to update the transmission statistics for a given rate.
- ath\_rate\_tx\_complete\_11n(): This function calls the rxRateUpdate\_ht() (for 11n) which updates the transmission statistics for a given rate.

Other functions include:

- `ath_rate_mapix()`: Maps the base index to a rate in the rate table
- `ath_rate_set_mcast_rate()`: Returns the rate used for multicast transmissions
- `ath_rate_table_init()`: Sets up the rate tables
- `ath_rate_attach()`: Initializes rate control, allocates memory and sets up the per STA tables
- `ath_rate_detach()`: Frees memory for the rate control per STA tables
- `ath_rate_create_vap()`: Allocates memory for Rate Control VAP structures
- `ath_rate_free_vap()`: Frees rate control VAP structures
- `ath_rate_node_alloc()`: Allocates memory for rate control node structures
- `ath_rate_node_free()`: Frees memory for rate control node structures
- `ath_rate_node_init()`: Initializes rate control node structures
- `ath_rate_node_cleanup()`: Cleans up rate control node structures
- `ath_rate_newassoc()`: Called when a new STA associates.
- `ath_rate_node_gettxrate()`: Returns current transmit rate to a STA
- `ath_rate_getmaxphyrate()`: Returns the maximum valid PHY rate for a STA
- `ath_rate_newstate()`: Called when the STA state changes (for example, associate to disassociate)
- `ath_rate_node_update()`: Rate control parameter update
- `ath_rate_findrateix()`: Returns the rate index, given a rate
- `ath_rate_max_tx_chains()`: Returns the maximum number of tx chains to be used for a rate, given the rate index

For offload design, the file `ar_wal_rc.h` and `ratectrl_if.h` defines all the rate related APIs that the rate control module exports. Refer to these sources for up-to-date descriptions of various APIs provided.

## 7.1.6 Configuration

There are no CLI commands to change the default rate control operations for direct attach devices

For offload devices, there are user level `iwpriv` APIs to enable/disable certain feature affecting rate control. These are:

- `set11NRates`: Disable auto rate when passed a valid HT MCS. When called with argument 0 turn on auto rate.
- `enable_rtscts`: Enable/disable use of RTS/CTS when first transmission attempt fails.
- `short_gi`: When disabled, the ratecontrol module does not use short gi at all. When enabled, the ratecontrol module use short gi for highest MCS.
- `stbc_tx`: enable/disable use of STBC for 1x1 rates
- `ldpc`: enable/disable use of LDPC

## 7.1.7 Rate Control for Special Traffic Types

The rate control algorithm has special features that are applied to special traffic types. In particular, there are special conditions applied to voice traffic and video traffic.

### 7.1.7.1 Rate Control for Voice traffic

The major source of voice traffic on WLAN networks are cell phones which have WLAN capability. However, very few phone/phone networks mark this traffic as voice using the TOS/DSCP bits in the IP header. Therefore rate control cannot identify voice traffic. However, most cell phones generally support UAPSD, so rate control uses this property to identify possible sources of voice traffic.

In contrast to static STAs, the channel quality of the link to a phone tends to vary more and at a faster rate because a cell phone may be in motion. To handle the fast varying link, special rules are applied to rate control:

- Only a subset of rates in the Rate Table are valid rates for transmission to a UAPSD STA. This ensures that the rate drops down much faster for voice traffic when there are packet errors.
- The retries in the rate series is set so that there are fewer retries at the higher rates in the series and a larger number of retries at the lowest rate. This also forces the rate to drop faster when the PER rises

### 7.1.7.2 Rate Control for Video traffic

If configured, rate control can apply a modified version of the rate control algorithm for video traffic. Refer to the *Video over Wireless* section for more details.

## 7.2 256 QAM rate support in 2.4 GHz operation

802.11ac protocol introduces 256-QAM modulation scheme in WLAN. This feature allows devices to support higher rate codes such as MCS8 and MCS9. These rates are higher than the max supported rates of traditional HT mode. This feature is proposed for 5GHz band only.

To achieve higher throughput in 2.4 GHz operating modes, this feature enables 256-QAM rates. As per 11n protocol, devices operating in 2.4 GHz band uses 64-QAM modulation to support max rates. With this feature, 256-QAM modulation technique would be enabled to support higher max rates.

### 7.2.1 256 QAM Rate Support Negotiation

256 QAM rate support feature adds VHT rates to the existing HT rate set in 11NG modes. The max supported rate with 256 QAM enabled are:

| NSS | LDPC   |        | BCC    |        |
|-----|--------|--------|--------|--------|
|     | 20 MHz | 40 MHz | 20 MHz | 40 MHz |
|     |        |        |        |        |

|   |         |         |         |         |
|---|---------|---------|---------|---------|
| 1 | MCS 8,9 | MCS 8,9 | MCS 8   | MCS 8,9 |
| 2 | MCS 8,9 | MCS 8,9 | MCS 8   | MCS 8,9 |
| 3 | MCS 8,9 | MCS 8,9 | MCS 8,9 | MCS 8,9 |

To indicate 256 QAM rates support, VHT IEs (capabilities & operation) are sent in the following management frames in 11NG mode. AP and Station exchange VHT IEs during association. The 256 QAM rates are enabled based on Rx/Tx rates map indicated in VHT capabilities IE.

- Beacon frame
- Association request
- Association response
- Reassociation request
- Reassociation response
- Probe request
- Probe response

## 7.2.2 Implementation

256 QAM rate support can be enabled if device is 11ac capable only. To enable/disable 256 QAM support in 2.4 GHz, a CLI command is invoked. Whenever users invoke the CLI command to enable 256 QAM, a flag “iv\_256qam” is set on VAP. This flag is checked while preparing/parsing VHT IEs in management frames.

On successful association, WMI command is issued to the firmware indicating capabilities and supported rates. Target will use the 256QAM rates set, if 256QAM rates are indicated via command “WMI\_PEER\_ASSOC\_CMDID”. The Node is marked as VHT capable node, if it successfully negotiates 256QAM rates.

## 7.2.3 CLI Commands

### Enable/Disable 256QAM support

| Command      | Usage                    | Description                                       |
|--------------|--------------------------|---|
| vht_11ng     | iwpriv athX vht_11ng     | It enables/disable 256QAM support                 |
| get_vht_11ng | iwpriv athX get_vht_11ng | It returns whether this support is enabled or not |

## 7.3 Disable Selected MCS For Given SSID

This features adds support to selectively disable specific HTMCS and VHTMCS rates for a VAP. The disabled HTMCS and VHTMCS rates are not used to transmit any frames to the nodes connected to that particular VAP. Other VAPs continues to use all supported rates.

### NOTE

- This feature cannot be used to disable any specific legacy rates.
- Any throughput related issues noticed after disabling few HTMCS and VHTMCS rates using this feature are not valid. Those issues need to be retested after removing these configurations.
- The associated STA can still try to send frames with disabled HTMCS and VHTMCS rates. The AP doesn't have control on that except advertising no Rx support for the disabled rates.
- The feature is valid only for AP VAPs.
- VHTMCS 0 to 7 cannot be disabled using this feature as there is a limitation in the protocol itself.
- VHT Capabilities IE does not have fine control to disable VHTMCS 0 to 7 in the Rx supported MCS set field.

### 7.3.1 Theory of Operation

To disable the specific MCS rates for a vap, there are two steps.

1. Stop advertising support for those specific MCS rates in the Beacon, Probe Resp and Assoc Resp frames. If we stop advertising support for those MCS rates, STA will try to avoid sending data packets with those rates.
2. When the STA connects with the AP, mask those specific disabled MCS rates and initialize the STA as if it doesn't support those specific rates, so that we don't transmit out data packets to STA with those specific rates.

By doing this way, we ensure that our rate control algorithm (rcFind) does not choose any of those masked MCS rates.

### 7.3.2 Implementation

#### 7.3.2.1 Stop advertising support for Specific MCS rates.

In umac, **ieee80211\_add\_htcap** function gets called to setup htcap IE in beacons, probe response and assoc response. **ieee80211\_add\_htcap\_cmn()** gets called from **ieee80211\_add\_htcap()** sets the list of supported Rx MCS rates by our driver through this function **ieee80211\_set\_htrates**.

Mask the disabled MCS rates inside this **ieee80211\_set\_htrates()**.

#### 7.3.2.2 Initializing STA (**ieee80211\_node**, **ath\_node**) with specific MCS rates disabled

In umac, when a new STA tries to join and sends the assoc req, we will use the information in the Association request to initialize the node's (ni) supported HT rates (ni\_htrates). This is done in the function **ieee80211\_setup\_ht\_rates** called from **ieee80211\_recv\_asreq**.

From **assoc\_ie->htcap->hc\_mcsset**, we will get to know what are all the MCS rates supported by STA. We will populate the ni→ni\_htrates based on htcap->mcsset and we need to mask the list of disabled MCS here and set only those allowed rates in ni->ni\_htrates.

Once the list of supported is derived, we will sort it in **ieee80211\_sort\_rate()** and we will also compute the list of support tx rates in **ieee80211\_compute\_tx\_rateset()** and both the supported tx rates by our vap and rx rates supported by the STA will be intersected in **ieee80211\_xsect\_rate()** and that will become the list of valid rates for the node.

This **ni->ni\_htypes** will be passed down later to lmac rate control and rate control will populate the list of valid ht rates for this node and stored in **ath\_node->an\_rc\_node**. **an\_rc\_node** is of structure type **atheros\_node** and this structure will hold the rate control information for the node used by lmac's rate control. Information in this structure will be used during **rcFind()** [ Rate find function].

Structure **atheros\_node** has a member field **txRateCtrl** of structure type **TX\_RATE\_CTRL** which has **validPhyRateIndex**, **validRateIndex** and **validRateSeries** which will be populated once during the init time.

During the init time, the members referred in **TX\_RATE\_CTRL** will get populated with the information **ni\_htypes** that we pass from umac and information populated in these lmac structure **an\_rc\_node** will be used extensively during rate find algorithm to choose a proper transmit rate for frames to transmit to the STA.

Since we have all already masked the **ni\_htypes** in umac and passing only allowed htates to lmac's rate control, the **ath\_node->an\_rc\_node** won't be populated with those disabled MCS rates and hence our AP can't choose the disabled MCS rates to transmit data frames out.

### 7.3.3 Offload Implementation

In case of offload design, host informs the valid HTMCS and VHTMCS through the **wmi\_peer\_assoc\_complete\_cmd** (**assoc\_cmd**). **assoc\_cmd->peer\_ht\_rates** carries informs about the valid HTMCS to the target rate control module. Since we have already masked the rates in **ni->ni\_htypes**, just populate the **ni\_htypes** info to **assoc\_cmd->peer\_ht\_rates**.

Similarly copy **ni->ni\_rx\_vhtypes** to **assoc\_md->peer\_vht\_rate->mcs->rx\_mcs\_set** and **ni->ni\_tx\_vhtypes** to **assoc\_md->peer\_vht\_rate->mcs->tx\_mcs\_set** to control the tx and rx mcs set of VHT rates.

### 7.3.4 CLI Guide

#### **iwpriv athX conf\_11acmcs configuration examples.**

```
iwpriv athX conf_11acmcs 0xFFFFFFFFC0
```

This command enables VHTMCS 0 – 7 in Spatial Streams 1,2,3 and says Spatial Streams (SS) 4 to 8 is not supported.

```
iwpriv athX conf_11acmcs 0xFFFFFFFFF5
```

This command enables VHTMCS 0 – 8 in Spatial Streams 1,2 and says Spatial Streams (SS) is not 3 to 8 supported.

```
iwpriv athX conf_11acmcs 0xFFFFFFFFFA
```

This command enables VHTMCS 0 – 9 in Spatial Streams 1,2 and says Spatial Streams (SS) is not 3 to 8 supported.

Only those bits are valid w.r.t the number of Tx and Rx spatial streams will be considered.

The LSB 16 bits passed to conf\_11acmcs represents the 16 bits in Rx MCS Map and Tx MCS Map field of VHT Capabilities IE. The MSB 16 bits should be 0xFFFF always. Please refer to the below picture on how to interpret the LSB bits of Rx MCS Map and Tx MCS Map fields of VHT Capabilities IE.

| B0                   | B1                   | B2                   | B3                   | B4                   | B5                   | B6                   | B7                   | B8 | B9 | B10 | B11 | B12 | B13 | B14 | B15 |
|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----|----|-----|-----|-----|-----|-----|-----|
| Max VHT-MCS For 1 SS | Max VHT-MCS For 2 SS | Max VHT-MCS For 3 SS | Max VHT-MCS For 4 SS | Max VHT-MCS For 5 SS | Max VHT-MCS For 6 SS | Max VHT-MCS For 7 SS | Max VHT-MCS For 8 SS |    |    |     |     |     |     |     |     |

Bits:      2      2      2      2      2      2      2      2      2      2      2      2      2      2      2

The Max VHT-MCS For n SS subfield (where n = 1,..., 8) is encoded as follows:

- 0 indicates support for VHT-MCS 0-7 for n spatial streams
- 1 indicates support for VHT-MCS 0-8 for n spatial streams
- 2 indicates support for VHT-MCS 0-9 for n spatial streams
- 3 indicates that n spatial streams is not supported

**NOTE** Iwpriv athX g\_conf\_11acmcs will display the configured value on the vap

#### Iwpriv athX disable11nmcs configuration examples.

Every bit in this command represents a MCS rate. LSB bit 0 represents and MSB bit 31 represents MCS 31. If a particular bit is set, then that MCS rate will be disabled for that vap.

```
iwpriv athX disable11nmcs 0xFFFFFFFF0
```

This command disables HTMCS 4 to 31 and enables only MCS 0 to 3.

```
iwpriv athX disable11nmcs 0xFFFF0FFF
```

This command enables MCS 12 to 15 and disables rest of the HTMCS rates.

**NOTE** Iwpriv athX g\_disable11nmcs will display the configured value on the vap

### 7.3.5 Validation

#### To validate the feature on Direct Attach design.

Disable specific MCS using the iwpriv command. Connect an 11n client to the AP and run some traffic.

It is nice to reset the stats on the radio once before running the traffic and checking the stats after running traffic.

Run the following command and check whether only those allowed mcs rates are exercised by checking the mcs counters stats

```
athstats -i wifi1 | grep mcs
```

#### To reset stats in Direct Attach radio:

```
athstatsclr -I wifi1
```

#### To validate the feature on Offload design

Disable specific MCS using the iwpriv command. Connect an 11ac client to the AP and run some traffic. It is nice to reset the stats on the radio once before running the traffic and checking the stats after running traffic.

Run the following command and check whether only those allowed mcs rates are exercised by checking the mcs counters stats

```
iwpriv athx txrx_fw_stats 6
```

#### To reset stats in offload radio

```
iwpriv athX txrx_fw_st_rst 0x3ff
```

## 7.4 Disable selected legacy rates for given SSID

### 7.4.1 Purpose

Many access points in close proximity, produce a significant amount of beacons using airtime and transmissions at lower rates take a lot of airtime. To decrease the airtime, AP should not transmit beacons and other management frames at lowest rate.

### 7.4.2 Overview

This feature adds support to disable the selected basic supported rates per VAP that user does not want to operate at. This feature is only applicable for the HOSTAP mode. The disabled rates are not used to transmit any frames to the stations connected to that AP. Different rates can be controlled on different VAPs via iwpriv commands.

### 7.4.3 Background theory

To disable the specific legacy rates for a VAP, following steps needed to be performed.

- When a user tries to disable some selected rates via iwpriv command, VAP in AP mode gets re-initiated with the corresponding rates getting masked for the VAP. While initiating the rates for the node, mask the rates to be disabled. Also choose the lowest available basic rate to be used as rate for the management frames and RTS / CTS frames.
- Stop advertising support for those specific legacy rates in the Beacon, probe response and association response frames.

- If AP stops advertising support for those rates, STA should not advertise the supported rates in the Association request frame.
- AP sends the management frames at the lowest available basic rate and STA sets its supported rates in the IE as per the AP's supported rates but the assoc request frame rate are 1 Mbps (2.4 GHz) and 6 Mbps (5 GHz).

#### 7.4.4 Implementation

##### Removing specific rates from node rates

In umac, based on the SSID a HOSTAP node will get created at first. While allocating the node we need to set the channel and default rate using **ieee80211\_node\_set\_chan**. The rates for the node is selected prior to the desired phy mode on the channel. As per the design after the rates are being initialized for the node, check the information of the node. If that node is a bss node only, then disable the selected rates from the node rates (ni\_rates) using **ieee80211\_disable\_legacy\_rates**.

In **ieee80211\_disable\_legacy\_rates**, mask the list of disabled legacy rates and setting only the allowed rates in ni->ni\_rates. Also, select the available lowest basic rate as **vap->iv\_mgt\_rate** which is used for management frames and RTS / CTS frames.

When a new STA tries to join and sends an Association request, use the information in the association request and **mask** the rates user wants to disable in **ieee80211\_setup\_rates** called from **ieee80211\_recv\_asreq**. Once the list of supported rates has derived, sort those rates using **ieee80211\_sort\_rate** and then do the intersection in between the rates supported by AP and STA using **ieee80211\_xsect\_rate**. Those rates are taken as the valid rates for the node.

This ni->ni\_rates is passed down later to lmac rate control and rate control populates the list of valid rates for this node and stored in ath\_node->an\_rc\_node. an\_rc\_node is of structure type atheros\_node and this structure holds the rate control information for the node used by lmac's rate control. Information in this structure is used during rcFind() [Rate find function].

Structure atheros\_node has a member field txRateCtrl of structure type TX\_RATE\_CTRL which has validPhyRateIndex, validRateIndex and validRateSeries which is populated once during the init time.

During the init time, the members referred in TX\_RATE\_CTRL gets populated with the information ni\_rates that pass from umac and information populated in these lmac structure **an\_rc\_node** is used extensively during rate find algorithm to choose a proper transmit rate for frames to transmit to the STA. Since the ni\_rates are masked in umac and passing only allowed rates to lmac's rate control, the ath\_node->an\_rc\_node is not populated with those disabled legacy rates and hence AP do not choose the disabled legacy rates to transmit any frames out.

Since the rates are masked in the umac, it has same impact in both DA and Offload implementation. Previously we had RTS / CTS rate as radio specific. Since this feature is implemented per VAP so introduced a new wmi command:

**WMI\_VDEV\_PARAM\_RTS\_FIXED\_RATE**

Host sends the rate code of the rate that is used for RTS / CTS frames and firmware sets the rate accordingly.

Choose the control (RTS / CTS) rate from the list of basic supported rates that are used for management frames.

#### **Stop advertising support for specific legacy rates**

In umac, while the beacon is getting initiated **ieee80211\_add\_rates** (supported rates) and **ieee80211\_add\_xrates** (extended supported rates) are getting called to setup supported rates information in beacons, probe response and assoc response.

Since the selected rates are masked, user wants to disable while initiating rates for the node itself. In beacon, probe response and association response, those rates are not advertised as supported rates to the STA.

### **7.4.5 Configuration**

There are 12 legacy rates and every bit in the value passed in iwpriv command represents a legacy rate. LSB bit 0 (B0) represents 1 Mbps and MSB bit (B11) represents 54 Mbps. If a particular bit is set, then that legacy rate are disabled for that VAP.

#### **IWPRIIV command**

`iwpriv athX dis_legacy 0x(value)`: Disables selected legacy rates for the VAP

`iwpriv athX g_dis_legacy`: Displays the configured value on the VAP

**NOTE** Do not disable all the basic supported rates.

#### **Example**

```
iwpriv athX dis_legacy 0x000f (Not allowed in 11b only)
iwpriv athX dis_legacy 0x015f (Not allowed in 11b, 11G, 11NG, 11A, 11NA,
11AC)
iwpriv athX dis_legacy 0x0150 (Not allowed in 11A, 11NA, 11AC)
```

### **7.4.6 Validation**

#### **Direct attach**

Disable specific legacy rates using the iwpriv athX dis\_legacy command. Connect an 802.11b/802.11g/802.11ng client to the AP and run some traffic. Check in the sniffer capture that beacon, probe response and association response, RTS /CTS frames are being transmitted at the lowest available basic rate. Also, the supported rates IE in beacon, probe response, assoc request, and assoc response must not contain any disabled rate.

For data frames, disable the highest available legacy rate and check. The following are the possible cases:

- In 802.11b mode, disable 11 Mbps rate and check whether the data frames are being transmitted at 5.5 Mbps or not.

- Similarly, in 802.11G mode, disable 54 Mbps and check whether the data frames are being transmitted at 48Mbps or not.
- This cannot be checked in 802.11NG mode because data frames cannot choose legacy rates in this mode.

### Offload

Disable specific legacy rates using the iwpriv athX dis\_legacy command. For a 5 GHz radio, connect an 802.11a /802.11na /802.11ac client to the AP; for a 2.4 GHz radio, connect an 802.11b/802.11g/802.11ng client to the AP and run a certain amount of traffic.

Check in the sniffer capture that beacon, probe response and association response, RTS /CTS frames are being transmitted at the lowest available basic rate. Also the supported rates IE in beacon, probe response, assoc request and assoc response should not have any disabled rate.

For data frames, disable the highest available legacy rate and check. The following are the cases:

- In 802.11b mode, disable 11 Mbps rate and check whether the data frames are being transmitted at 5.5 Mbps or not.
- Similarly, in 802.11G and 802.11A mode, disable 54 Mbps and check whether the data frames are being transmitted at 48 Mbps or not.
- This cannot be checked in 802.11NG/ 802.11NA/802.11AC mode, since data frames cannot choose legacy rates in these modes.

### 7.4.7 Limitation

- This feature is limited to disable only the basic supported rates but user should not disable all the basic supported rates at once.
- This feature is not applicable for HT rates.
- This feature is valid only for AP VAPs.

## 7.5 Configure short guard interval for all MCSs

By default, the very high throughput (VHT) short guard interval (SGI) mask is enabled for all modulation coding schemes (MCSs). Enter the iwpriv athX vht\_sgimask <value> command to specify the VHT SGI mask for a specific MCS. For example:

- A value of 0x4 indicates VHT SGI mask set for only MCS2
- A value of 0xFF indicates VHT SGI mask set for MCS0-MCS7
- A value of 0x3FF indicates VHT SGI mask set for MCS0-MCS9

Enter the iwpriv athX get\_vht\_sgimask command to retrieve and display the configured SGI mask for the corresponding MCSs.

# 8 Power Management Techniques

---

This chapter describes the power management mechanisms, such as UAPSD WMM Power Save (WMM-PS), Transmit Power Control (TPC), Advanced Enterprise for WLAN driver version 10.4, and thermal mitigation.

## 8.1 UAPSD WMM Power Save (WMM-PS)

WMM Unscheduled Automatic Power Save Delivery (UAPSD) is a power save mechanism specified by the WiFi Alliance WMM specification. UAPSD mechanism applies to unicast QoS Data and QoS Null data frames only. This mechanism provides quicker way to retrieve the buffered frames in the AP for a STA in power save mode. This section describes the UAPSD operation and implementation details. Refer to the WMM specification document for more details on UAPSD operation.

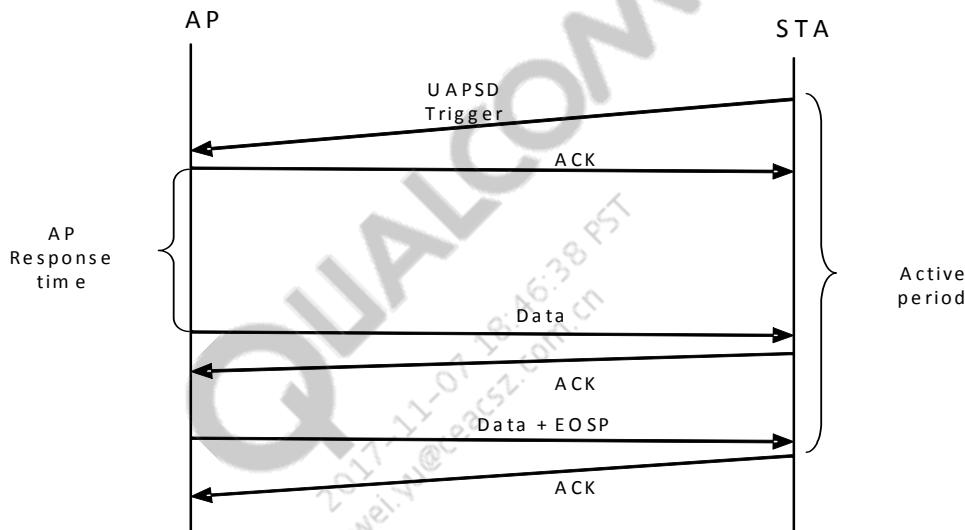
### 8.1.1 Terminology

- AC (Access category): A label for the common set of enhanced distributed channel access (EDCA) parameters that are used by a WMM STA to contend for the channel in order to transmit MSDUs with certain priorities. WMM defines 4 ACs.
- Delivery-enabled AC: An AC for a specific STA, to deliver traffic in that STA-specific AC using WMM when an Unscheduled Service Period (USP) is triggered by that STA.
- Service Period (SP): A service period is a contiguous time during which one or more downlink unicast frames are transmitted to a WMM STA and/or one or more TXOPs are granted to the same WMM STA. Service Periods can be scheduled or unscheduled. For a WMM STA, there can be at most one Service Period active at any time.
- Trigger-enabled AC: An AC for a specific STA to initiate an Unscheduled Service Period (USP), if one is not already in progress, when frames are received from that STA are of subtype QoS Data or QoS Null associated with that AC.
- Trigger Frame: A QoS Data or QoS Null frame from a WMM STA in Power Save Mode associated with an AC the WMM STA has configured to be a trigger-enabled AC. A QoS Data or QoS Null frame that indicates transition to/from Power Save Mode is not considered to be a Trigger Frame and the AP does not respond with a QoS Null frame.
- Unscheduled Service Period (USP): The Service Period that is started when a WMM STA transmits a trigger frame to the WMM AP.

## 8.1.2 Theory of Operation

The UAPSD mechanism used by the STA retrieves unicast QoS traffic buffered in the AP by sending trigger frames. During association/re-association, a STA indicates (in the QoS Info field) which Access Categories are UAPSD enabled. For non-UAPSD enabled ACs, the STA shall use PS-Polls as a means to retrieve the legacy power-save buffered frames.

For a UAPSD-enabled AC, the AP responds to a trigger by sending up to a maximum service period (SP) number of frames. The last QoS data frame shall have the End of Service Period (EOSP) bit set to indicate the end of the service period. If the AP has no buffered frames for any delivery-enabled ACs, it responds with a QoS NULL frame with the EOSP bit set. [Figure 8-1](#) illustrates a typical exchange between the STA and AP.



**Figure 8-1 UAPSD Mechanism**

The AP is compliant with the WMM Power Save Specification and is required to pass the WMM Power Save certification.

## 8.1.3 WMM Power-Save Advertisement

The UAPSD-enabled AP advertises its capability by setting the UAPSD (bit 7) of the QoS Info field in either the WMM Information Element or the WMM Parameter Element. This capability is advertised in beacons, probe responses, and (re)association responses.

## 8.1.4 U-APSD Enabled STA Association

The QoS Info field in the association request from the STA indicates which ACs will use the UAPSD mechanism. It also indicates the maximum SP length, which indicates the number of buffered frames the AP may send per USP. The AP shall maintain this per node information in its data structure. The current implementation does not support TSPEC; therefore, these parameters can only be negotiated during (re)association time.

### 8.1.5 STA and Power Management

The STA sets the PM bit in the frame control field to indicate that it is in power-save mode. The AP buffers frames of delivery-enabled ACs in the UAPSD queue and non-delivery enabled ACs in the legacy power-save queue.

The AP does not buffer frames for a STA in active mode, as indicated by a 0 in the PM bit.

### 8.1.6 STA and UAPSD Trigger

The STA in power-save mode can send a QoS NULL or QoS Data frame to trigger the AP to send buffered frames. The AP shall acknowledge the trigger and then follow it up with up to maxSP number of frames. The last frame transmitted in the SP shall have the EOSP bit set. All transmitted frames shall have the MORE bit set with the exception of the last frame that has the EOSP set. The last frame shall have the MORE bit set only if there are more buffered frames in the delivery enabled queue.

If there are no buffered frames on the AP when the trigger is received, the AP responds with a QoS NULL frame with EOSP set to end the current service period.

To improve the power consumption performance of the STA, the AP must respond to triggers expediently.

### 8.1.7 Implementation

The implementation of UAPSD involves two major components.

1. Queuing the QoS frames in UAPSD queue when STA is in UAPSD power save
2. Processing UAPSD triggers

[Figure 8-2](#) explains these two major components of UAPSD processing.

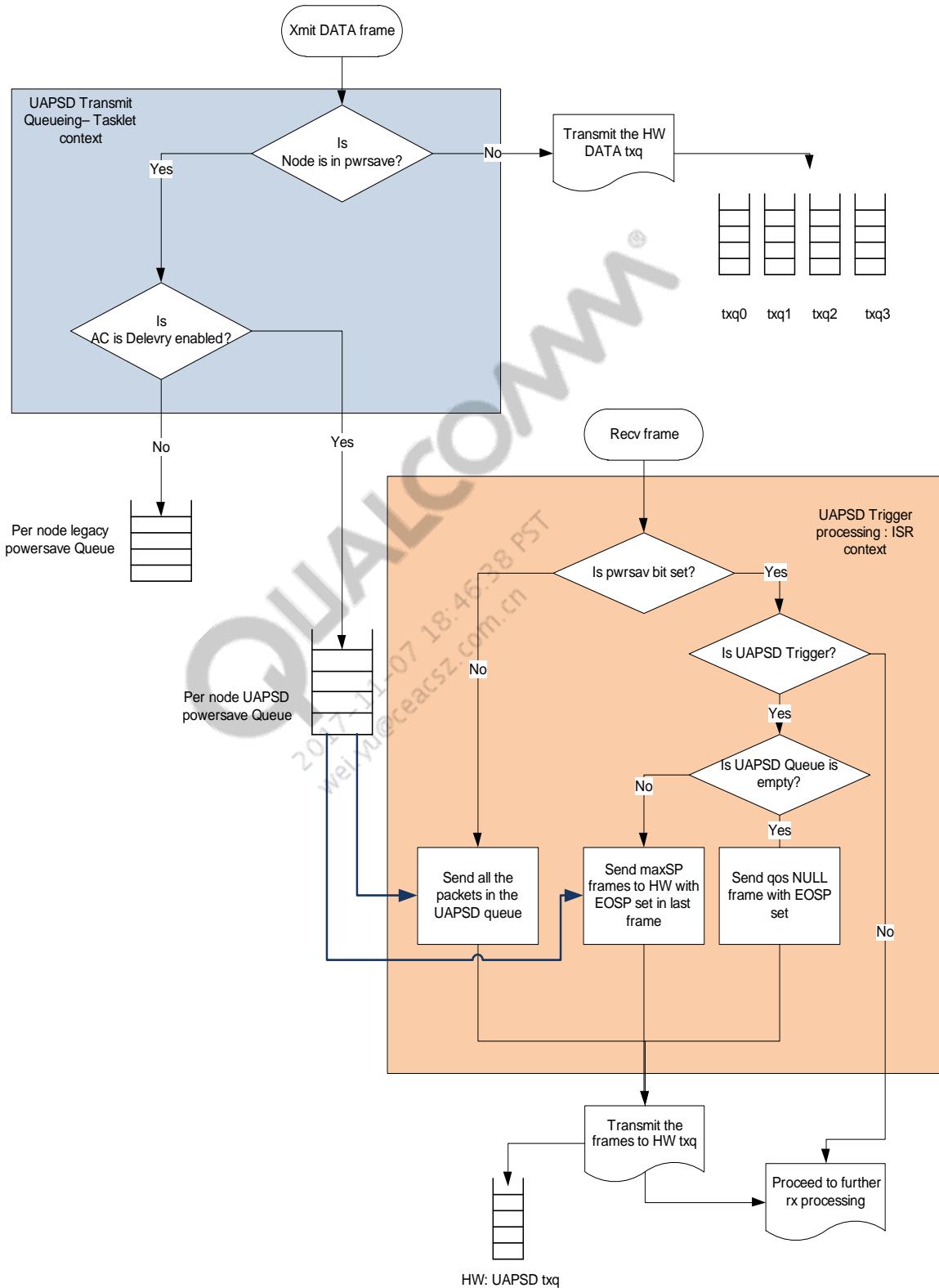


Figure 8-2 UAPSD Processing

### 8.1.7.1 Queuing of UAPSD frames

For each QOS DATA frame, if the UAPSD is supported then the following two checks are done

1. Is the power save enabled for that node?
2. Is the UAPSD AC flag for the station node delivery enabled corresponding to the DATA frame AC?

If both conditions are met, then the frame gets queued in the per-node UAPSD transmit queue. If the first check alone met then the data frame gets queued in the per-node legacy power save queue.

### 8.1.7.2 UAPSD Trigger processing

In most of the AP platforms, UAPSD trigger processing is done in ISR context for a faster UAPSD response to trigger frames. A separate hardware txq is used to transmit the UAPSD buffered frames. The UAPSD hardware queue allocated with higher priority compared to the four DATA hardware queues corresponds to each AC. The separate high priority hardware queue assigned to UAPSD allows the buffered frames to be transmitted ahead of any pending data frames in the hardware data queues. This further helps in minimizing the response time for UAPSD triggers. The UASPD hardware queue is configured with a bursting option enabled so that UASPD frames for each service period gets transmitted in burst with SIFS interval.

### 8.1.7.3 Code base

The changes for UAPSD implementation are done in both the UMAC and LMAC layers.

### 8.1.7.4 UMAC modifications

The UMAC modifications are largely performed in the following modules

- txrx – UMAC transmit and receive module
- if\_lmac – UMAC interface to LMAC modules

#### **txrx changes**

**ieee80211\_output.c:** The transmit entry function in the file “wlan\_vap\_send” is changed to check for a UAPSD delivery-enabled flag for node and AC corresponding to the frame that needs to be transmitted. If the flag is enabled, then legacy power-save queuing is avoided for these frames.

#### **if\_lmac changes**

**if\_ath\_uapsd.c:** New file added to handle the UMAC UAPSD processing. The main functionalities include the following:

- UAPSD triggers verification
- QOS NULL frame allocation
- Clearing of service period flag at the end of service period frame completion.

**if\_ath.c:** The following functionalities are modified for UAPSD processing.

- **ath\_tx\_send:** The if\_lmac module that transmits the entry function is modified to mark the wbuf as a UAPSD buffer if the frame is identified as a UAPSD frame. The TIM bit and MORE bit are set.
- **ath\_tx\_prepare:** This function is modified to set the uapsd flag in the “txctl” structure when the wbuf is marked with a UAPSD flag.

### 8.1.7.5 LMAC modifications

All the UAPSD changes in LMAC are in the ath\_dev module.

#### **ath\_xmit.c changes**

**ath\_tx\_start\_dma:** Modified to queue the frame to per-node UAPSD queue when the UAPSD flag is set in the txctl structure.

#### **ath\_recv.c changes**

**ath\_rx\_proc\_descfast:** The rx frame processing in ISR context calls a UMAC interface function to check and process the UAPSD trigger for the received frames.

**ath\_rx\_tasklet:** The rx frame processing in the tasklet context calls a UMAC interface function to check and process the UAPSD trigger for the received frames – enabled only when the UAPSD trigger processing in ISR context is disabled.

**ath\_uspad.c:** New file added in ath\_dev module to implement the LMAC changes related to UAPSD only. The functionalities included in this file are

- Implementation of per-node UAPSD software queue.
- Setting up the UAPSD hardware queue for transmission buffers which are stored in the UAPSD software queue.
- Transmission of frames from UAPSD software queue once the UAPSD trigger is identified by the UMAC interface function.
- Transmission completion handling of UAPSD frames. Completion of UAPSD frames with EOSP is set, and is indicated to the UMAC interface function to update the service period completion.

## 8.2 Power Management

Since the majority of wireless devices are mobility based, it is important to conserve power. The IEEE 802.11 specification proposes a method in which the wireless stations (STA) conserve power by means of exchanging power management state information with the associated Access Point (AP).

### 8.2.1 Terminology

| Term                                | Description  |
|-------------------------------------|--|
| Power management field              | This is a one-bit field in the frame control. It is used by the STA to indicate its power management state to the AP. If enabled, the STA goes into power save mode and sleeps. If disabled then the STA is in the awake state.  |
| Traffic Indication Message          | The TIM information element is present only within the beacon frames generated by AP. It indicates the presence of buffered data to the associated STAs.   |
| Delivery Traffic Indication Message | The DTIM is a kind of TIM that informs the STA about the presence of buffered multicast/broadcast frames at the AP. It is generated within the periodic beacon at a frequency specified by DTIM interval.  |
| Listen Interval                     | The listen interval is used by the STA to indicate to an AP how often a STA in power save mode wakes to listen to beacon frames. The value of this parameter is expressed in units of beacon interval. The AP may use this listen interval information in determining the lifetime of frames that is buffered for a STA. |
| Awake Power State                   | A power state in which the STA is fully powered.   |
| Doze                                | A power state in which the STA is not able to transmit or receive frames; it consumes very low power.  |
| Active Mode                         | In this mode STA receives frames at any time. In active mode, a STA is in the awake state.   |
| Power save Mode                     | In this mode STA listens to selected beacon frames, based on the listen interval and sends PS-Poll frames to AP.   |

### 8.2.2 Theory of Operation

Only a STA can be in one of the two power modes; that is, the active mode when the STA can receive and send any frames at any time, and power save mode when the STA is mainly in a sleep state and transitions to the full powered state.

In the power save mode, all nodes in the network are synchronized to wake up periodically to listen to beacon messages. Broadcast/Multicast messages or unicast messages to a power saving STA are first buffered at the transmitter and announced via the Traffic Indication Map. The TIM information is present in the AP beacon frames. Broadcast/multicast frames are exchanged after the delivery of DTIM interval.

A STA changing the Power Management (PM) mode informs the AP using the PM bit of the frame control field of the transmitted frames. The AP will not transmit frames to STA if it is in power save mode; instead it will buffer all the frames and indicate its presence via the TIM. The STA periodically wakes up and listens to the TIM information element present in the beacon frame to check if there is any data buffered at the AP; this is dictated by the STA listen interval, which is negotiated during the association.

If any STA is in power save mode, the AP buffers all broadcast/multicast frames and delivers them to all STAs immediately following the next beacon frame containing the DTIM transmission. A STA remains in its current PM mode until it informs the AP of the PM mode change via a frame exchange that includes an acknowledgment from the AP.

### 8.2.3 Direct-attach Implementation

The power save feature for direct-attach devices is implemented in two logically separate parts, the one pertaining to AP and the other pertaining to STA.

#### 8.2.3.1 Source Code Base

- `ieee80211_power_priv.h`  
Contains defines that are private for the Power Management feature
- `ieee80211_power.c`  
Contains functions related to the Power Management feature
- `ieee80211_power_queue.c`  
Contains functions related to managing the frames meant for STAs in power save mode
- `ieee80211_sta_power.c`  
Contains functions related to the power management feature for STA implementation
- `ieee80211_ap_power.c`  
Contains functions related to the power management feature for AP implementation
- `Ieee80211_sta_power_private.h`  
Contains defines meant for the STA related power management feature
- `ieee80211_sta_power_smpls.c`  
Contains functions related to the power management state machine related STA implementation

**NOTE** STA side power management is not included in this document.

#### 8.2.3.2 Compile Flags

- `UMAC_SUPPORT_AP_POWERSAVE`  
Enables the AP-related power save feature
- `UMAC_SUPPORT_STA_POWERSAVE`  
Enables the STA-related power save feature
- `UMAC_SUPPORT_STA_SMPS`  
Enables the MIMO-related power save feature

- UMAC\_SUPPORT\_POWERSAVE\_QUEUE
  - Enables the power save queue feature

### 8.2.3.3 Data Structure

The node (ieee80211\_node) is the main data structure that contains the power management related information. Following are the members listed:

- Power save mode state (enabled/disabled)
- QoS enabled
- U-APSD power save state
- U-APSD triggerable state
- U-APSD SP in progress
- Association ID
- U-APSD per node flags
- U-APSD Max SP
- U-APSD AC Trigger enabled for each AC
- U-APSD AC Delivery enabled for each AC

### 8.2.3.4 Software APIs

This section details the available software APIs for the AP power save feature.

#### File: ieee80211\_ap\_power.c

- Function: ieee80211\_ap\_power\_vattach

This function initializes power management related parameters meant for the VAP interface in the access point. The function allocates memory for the TIM bit map and performs TIM-related initialization. Called during the VAP attach.

- Function: ieee80211\_ap\_power\_detach

This function cleans up power management related parameters meant for the VAP interface in the access point. Free up the allocated memory for TIM bit map. Called during VAP detach.

- Function: ieee80211\_set\_tim

This function sets or clears the TIM bit map for the given node.

- Function: ieee80211\_power\_alloc\_tim\_bitmap

This function allocates memory for the TIM bit map information.

#### File: ieee80211\_power.c

- Function: ieee80211\_power\_arbiter\_vattach

This function initializes power-arbiter related fields for the given VAP interface. Called during the VAP attach.

- Function: ieee80211\_power\_arbiter\_vdetach

This function cleans up the power-arbiter related fields for the given VAP interface. Called during the VAP detach.

- Function: ieee80211\_power\_attach

Dummy initialization routine.

- Function: ieee80211\_power\_detach

Dummy cleanup routine.

- Function: ieee80211\_power\_vattach

This function initializes the power management related parameters called during VAP attach. Initialization is performed depending upon the current operating mode. Called during VAP attach.

- Function: ieee80211\_power\_detach

This function cleans up the power management related parameters. Called during VAP detach.

- Function: ieee80211\_set\_uapsd\_flags

This function sets the UAPSD flags.

- Function: ieee80211\_get\_uapsd\_flags

This function gets the current UAPSD flags.

- Function: ieee80211\_set\_wmm\_power\_save

This function sets the WMM power save state.

- Function: ieee80211\_get\_wmm\_power\_save

This function gets the current WMM power save state.

- Function: \_\_ieee80211\_power\_enter\_nwsleep

This function is called by modules that want to enter the network sleep mode.

- Function: \_\_ieee80211\_power\_exit\_nwsleep

This function is called by modules that want to exit the network sleep mode.

- Function: ieee80211\_power\_vap\_event\_handler

This function handles all power management related events for the given VAP interface.

#### File: **ieee80211\_power\_queue.c**

- Function: ieee80211\_node\_saveq\_drain

This function clears any frame queued on the node's power save queue and returns the number of frames that were present.

- Function: ieee80211\_node\_save\_age

This function ages the frames on the power save queue. The aging interval is four times the listen interval specified by the station.

- Function: ieee80211\_node\_saveq\_handle\_ps\_frames  
This function handles queuing of frames for power save.
- Function: ieee80211\_node\_saveq\_queue  
This function saves outbound packet for a node in power-save sleep state. The new packet is placed on the node's saved queue, and the TIM is changed if necessary.
- Function: ieee80211\_node\_saveq\_flush  
This function flushes the power save queue for the given node.
- Function: ieee80211\_node\_saveq\_send  
This function send one frame out of the power save queue, it is called in response to a PS-Poll frame.
- Function: ieee80211\_node\_saveq\_cleanup  
This function cleans up the power save queue for the given node. This is called when the node leaves the BSS.
- Function: ieee80211\_node\_saveq\_get\_info  
This function gets the power save queue info for the given node.
- Function: ieee80211\_node\_saveq\_attach  
This function initializes the power save-related queues for the given node. This is called when the node info is created.
- Function: ieee80211\_node\_saveq\_detach  
This function cleans up the power save-related queues for the given node. This is called when the info is deleted.
- Function: ieee80211\_node\_saveq\_set\_param  
This function sets the power save queue parameters such as the queue maximum length.

## 8.2.4 Offload Implementation

The power save feature for devices with offload architecture is split between the host driver and target firmware. The host is responsible for selecting and advertising the BSS policy and determine the final configuration of each associated STA. The firmware is responsible for monitoring the peer STA power management state, pausing TX queues when the STA is asleep, un-pausing when the STA is awake, Content After Beacon (CAB), TIM management, and sending downlink frames in response to PS-Poll frame and U-APSD triggers frames.

### 8.2.4.1 Host Driver Implementation

#### BSS power save policy

An AP is required to support the basic STA power save modes: PSnonPoll and PS-Poll. Optional U-APSD support may be advertised by setting the U-APSD bit in the QoS Info field of the WMM

parameter (or information) element. The WMM element is included in the AP Beacons and Probe Responses.

## STA association and determination of Legacy and U-APSD configuration

Legacy and U-APSD configuration is determined by the QOS info field in the WMM information element included in the association request sent STA to the AP duration association. The host UMAC parses the WMM information element to determine the STA power save configuration. After association is complete, this configuration is passed down to the firmware through the WMI.

### 8.2.4.2 Target Firmware Implementation

#### STA power save state management

The target firmware monitors the power management (PM) bit in all MPDU received from the associated STA. When the PM bit is zero, the STA is awake. When the STA enables power save, it will set the PM bit to one. When the firmware detects this power save state change, it pauses the STA Tx queues and begins buffering the downlink MPDUs. The STA Tx queues within the firmware are un-paused, and any buffered traffic is delivered when the firmware sees an uplink data frame from the STA with the PM bit value of zero; that is, the STA is now awake.

#### PS-Poll and U-APSD

The sleeping STA may send a PS-Poll frame to retrieve one buffered MPDU from the AP. The sleeping STA may also send a U-APSD trigger frame to start an unscheduled service period. The AP may transmit a MPDU on a delivery-enabled AC during this unscheduled service period to the STA. The number of MPDUs is bounded by the availability of buffered MPDUs and the STA maximum service period length as negotiated in the STA association request. If the STA starts an unscheduled service period and there are no deliver-enabled MPDUs, the AP sends a QoS-data-null with the EOSP to the STA.

These PS-Poll and U-APSD protocols are handled entirely by the target firmware with no additional host driver interaction, other than the normal Ts MSDU processing.

#### TIM management

The AP buffers MPDUs for the (TIM) in the AP beacon. At each software beacon announcement (SWBA), the target firmware sends a copy of the most up-to-date TIM information to the host driver. The host driver uses this TIM information to construct the TIM information element included in the beacon. The beacon is subsequently sent back to down to the firmware for transmission at the TBTT.

#### Content After Beacon (CAB)

The AP must buffer all multicast and broadcast data when one or more associated STA are sleeping. The buffered multicast and broadcast data are advertised in the TIM element and transmitted only after DTIM beacons. This multicast and broadcast data is called Content After Beacon (CAB) and the corresponding Tx queue is called the CAB queue. The target firmware manages the CAB queue, pausing it based on the power save state of all associated STAs.

#### 8.2.4.3 Firmware API (WMI)

The U-APSD configuration is the only power save configuration pushed down to the firmware for each associated STA (see WMI descriptions for WMI\_AP\_PS\_PEER\_PARAM\_CMDID, WMI\_AP\_PS\_PEER\_PARAM\_UAPSD, and WMI\_AP\_PS\_PEER\_PARAM\_MAX\_SP).

### 8.3 Transmit Power Control (TPC)

Target Power Control (TPC) refers to the mechanism used in determining the optimal power level to be used for packet transmission while ensuring that it satisfies all IEEE specification requirements, regulatory limits, and board limits.

Control over transmit power is desired on WLAN devices to improve SNR (and hence the spectral efficiency), and to minimize interference in the WLAN network, especially when there are other networks in the range, and to meet the regulatory limits.

The AR93xx series of chipsets supports TPC using two methods

1. Register based (rate-to-power mapping in PHY registers)
2. Per-packet TPC (descriptor based)

This section describes the support for TPC in software using the above methods. The scope of this description is limited to the AR93xx series of chipsets and 11ac Wave 1 QCA98xx chipsets. This algorithm applies to designs that did not take into account of TXBF/# of TX/# of SS/Antenna Gain/Array gain in CTL design/table. These designs includes all 11n Solution and First 11ac Wave 1 Solutions. All 11ac Wave 2 QCA99xx Solutions and later does not use this algorithm.

#### Acronyms

TPC: Transmit Power Control

TxBF: Transmit Beam Forming

STBC: Space Time Block Coding

CDD: Cyclic Delay Diversity

CCK: Complementary Code Keying

OFDM: Orthogonal Frequency Division Multiplexing

HT: High Throughput

FCC: Federal Communications Commission

ETSI: European Telecommunications Standards Institute

MKK: Japan Regulatory domain

## 8.3.1 Theory of Operation

Transmit Power Control can be enabled to work with the register method or the per-packet method in hardware. The per-packet method is chosen to be the default mode of operation.

### 8.3.1.1 Register-based method

Transmit power control is based on a set of “rate-to-power” tables programmed by the driver into an array of PHY registers. The “rate-to-power” array includes CCK, OFDM, HT20 and HT40 rates. During transmission, the hardware will use the target power value corresponding to the rate at which the packet is being transmitted from these registers.

The driver accounts for the following when programming these registers.

1. The hardware capability in terms of supported per-power rate.
2. The confirmatory test limits (CTL) per channel.
3. The regulatory power limitations.

The registers are programmed with an assumption on the maximum number of transmit chains configured and cannot support dynamic configuration of Tx chains. Note that limits 1) and 2) above are programmed into the EEPROM while 3) is hard coded in the driver per regulatory domain.

### 8.3.1.2 Per-packet method

This method is enabled by setting the TPC enable bit in the PHY power control register (PHY\_POWER\_TX\_RATE\_MAX). The power levels to be used with each of the rate series in the packet is specified in the descriptor at the time of transmit. The hardware will transmit the packet at the rate specified for the packet. Note that this method provides much more flexibility than the register based method.

### 8.3.1.3 TPC for self-generated frames

The hardware generates control frames (CTS, ACK, RTS) and for such frames the transmit power is determined from the value programmed in the Tx power control register (MAC\_PCU\_TPC). Per-packet TPC and register-based TPC methods described above are not applicable to these frames.

## 8.3.2 Implementation

TPC requirements may be different for different regulatory domains and the goal of this implementation is to make it flexible to accommodate the differences in requirements while at the same time meeting the performance criterion.

**NOTE** The algorithms described in this section apply to legacy designs that did not use the current CTL table design to capture separate compliant power values for transmit beamforming, number of transmit chains and array gain for number of spatial streams. These legacy designs includes all 11n and 11ac Wave 1 Solutions. All 11ac Wave 2

Solutions do not use the below algorithms since the latest CTL table design captures all required power values. In other words, 11ac Wave 2 and later solutions do not use run time SW calculations to determine regulatory compliant powers. Instead manufacturers capture all required regulatory powers during end-product conformance testing and enter the data in the CTL file stored in the BDF. Additionally, Qualcomm provides a scripting tool to assist in generation of the CTL file.

The power calculation algorithm factors in the following parameters.

- **CtryCode:** Country code for determining the regulatory domain.
- **MaxRegLimit:** Regulatory limit from the software regulatory domain table. For FCC this maps to the conducted power. For other domains this is the EIRP.
- **MaxAntenna:** Max allowed antenna gain from software regulatory domain table.
- **AntGain:** Actual antenna gain from the EEPROM. Should be factored in when computing the upper regulatory limit.
- **TxCPwr:** Multi chain gain (hard coded)
  - For 2 chains  $10 \cdot \log_{10}(2) = 3$
  - For 3 chains  $10 \cdot \log_{10}(3) = 4.5$
- **CtlEdgPwr:** Conformance test limits (CTLs) stored in EEPROM per hardware unit. These limits are an upper bound to be applied to the target powers. These limits ensure compliance to the regulatory certification test results specific to this product or module. The CTLs are stored independently per regulatory domain (FCC, ETSI and MKK) as the rules differ with regards to transmit power limits in the various bands.
- **TgtPwr:** The target powers stored in EEPROM (or on-chip) per hardware unit. These target powers are stored for each PHY rate and represent the hardware capability in terms of IEEE mask compliance (typically limiting at low rates) and IEEE Tx EVM requirements (typically limiting at highest rates).
- **PwrLimit:** User configured power limit. By default this is set at 63 (specified in 0.5 steps)
- **TxBFGain** – Coherent array gain with TxBF (see section on regulatory rules and limits). The back off values is hard coded in the driver.
- **STBCGain** – Coherent array gain with STBC (see section on regulatory rules and limits). The back off values is hardcoded in the driver.
- **CDDGain** – Coherent array gain with CDD (see section on regulatory rules and limits). The back off values is hardcoded in the driver.

### 8.3.2.1 Algorithm

**NOTE** The algorithms described in this section apply to legacy designs (11n and 11ac Wave 1 Solutions). 11ac Wave 2 and later solutions do not use the below algorithms since the latest CTL table design captures all required power values. Qualcomm provides a scripting tool to assist in generation of the CTL file.

The algorithm for computing target power per rate is as follows

1. The Antenna gain is read from EEPROM and if it is greater than the max allowed antenna gain (example: 6dBi in the case of FCC) defined for the regulatory domain, the difference between their values is subtracted from the maximum regulatory limit. If not the maximum limit is left unaltered.

scaledPower = MaxRegLimit + ((min (MaxAntenna - AntGain), 0)

2. The scaledPower from 1) is compared with the user configured power limit (PwrLimit) and the minimum of the two is chosen for further calculations.

scaledPower = Min (PwrLimit, scaledPower)

3. The scaled power is further reduced by a multi-chain gain factor (see TxCpwr above).

scaledPower = scaledPower - TxcPwr

4. The CTL for the regulatory domain is read from the EEPROM and if the current channel is determined to be a band edge in the CTL, the band edge maximum power is factored in. The minimum of the scaledPower (from step 3) and EdgePwr is used for per rate target power calculations.

minCtlPower = min (scaledPower, CtlEdgePwr)

5. The per rate target power read from the EEPROM is adjusted based on the control mode.

perRateTargetPower = min (TgtPower, minCtlPower)

**NOTE** There are adjustments made to the target power with BT Coexistence or PAPRD enabled. There is also support for specifying an additional power scale factor (TPC scaling). These exceptions are not of much significance to this discussion.

## Register based-method

The register-based method will be programmed with power values computed assuming CDD/Direct map mode of operation.

Because ETSI and MKK do not require the array gain be factored for CDD, the final **perRateTargetPower**, which is the minimum target power, band edge maximum power, and regulatory power limit for the current channel, is written to the group of power array registers. The FCC requires CDD to include coherent array gain and therefore the driver will adjust the **perRateTargetPower** by applying the **CDDGain** as specified by the FCC rules and the adjusted values are written into the power array registers.

## Per-packet method

Per-packet TPC is desired for finer control of target power per rate. With per packet TPC enabled, the driver dynamically adapts the target power based on the transmit mode (**CDD/Direct map**, **STBC or TxBF**). To enable easy lookup during transmit, the driver maintains three target power tables, one for each mode computed at the time of channel change/reset.

**Transmit Beam Forming:** FCC, ETSI, and MKK require that the array gain specified below be factored in with TxBF enabled. Note that in this case the rule applies only to **[N TX / ((N-1) streams]** configurations. The algorithm for computing **perRateTargetPower** described in the previous section will deduct the coherent array gain from the upper limit and adjust the per-rate

TPC to be within limits. For FCC, note that only the amount by which the sum of antenna gain and array gain exceeds the maximum allowed antenna gain is deducted.

- 2Tx/1-stream – 3 dB
- 3Tx/2-stream – 4.8 dB (rounded off to 5)
- 3Tx/1-stream –4.8 dB (rounded off to 5)

**Cyclic Delay Diversity:** The FCC requires CDD to include array gain and therefore the driver will treat CDD and TxBF modes the same for TPC calculations. ETSI and MKK do not explicitly require CDD to include the array gain.

**Space Time Block Coding:** The FCC requires STBC to factor in the following array gain. With STBC the driver will deduct only the amount by which the (Antenna Gain + Array Gain) exceeds the maximum allowed antenna gain from the upper limit and adjust the per-rate TPC to be within limits.

- 3Tx/1-stream – 4.8 dB (rounded off to 5)
- 3Tx/1-stream –4.8 dB (rounded off to 5)
- ETSI and MKK do not have this requirement.

Regardless of whether TxBF is ON or OFF, the algorithm will make sure that the best target power is used with each rate. This is to say that when TxBF is ON, if the target power computed without TxBF is within the regulatory limit factoring in the array gain, the target power will not be adjusted or reduced. The idea here is to maximize the power output while still meeting the regulatory limits.

On high power reference designs TxBF will be disabled if the per rate TPC for CDD is constrained by the upper regulatory limits (only applicable to ETSI and MKK).

The thought here is to avoid the full  $10 \cdot \log_{10}(N)$  conducted power reduction with TxBF. A table with ***DisableTxBF flag*** per rate is populated and maintained by the driver at the time of channel change/reset based on the TPC values, and this table is checked by the ***ratectrl*** logic before enabling TxBF for any packet before transmission.

All computations in the driver assume the power limits and antenna gain to be specified in steps of 0.5.

ETSI has different limits for the STA and AP when operating in the UNII-II extension band. This is handled dynamically in the driver based on the mode of operation (AP, STA) of the device.

### 8.3.2.2 Codebase

This section explains the various files that make up the TPC implementation. Most of the implementation is in the HAL while a small set of changes are present in the LMAC.

- Regulatory domain definitions: All regulatory domain-related information (country code, limits and such) is defined in the file *ah\_regdomain\_common.h*.
- TPC Power calculations: The logic to read power tables from EEPROM and the regulatory domain table, and compute the per-rate target power without factoring in the array gain are present in the file *ar9300\_eeprom.c*.

A separate table is maintained for each transmit mode (CDD, TxBF, STBC) for easy lookup (ar9300.h). The tables are defined as *txPower*, *txPower\_txbf* and *txPower\_stbc*.

TPC table generation functions factoring the array gain for CDD, STBC and TxBF are defined in the file ar9300\_phy.c. This file also contains the logic for disabling TxBF based on CDD power levels.

- TxBF Changes: A new table has been defined (*sc\_txbf\_disable\_flag*) in the LMAC (*ratectrl.h*) to allow easy lookup of TxBF status (ON/OFF) during packet transmission (files *ratectrl\_11n.c*, *ratectrl\_11nViVo.c*, *if\_athrate.c*). The table is initialized at the time of channel change/reset (*ath\_main.c*)
- Transmit path changes: During transmit with the per-packet method the per-rate power value is extracted from the appropriate table (CDD, TxBF, and STBC) based on the transmit mode and filled in the Tx descriptor for each of the rate series.

### 8.3.2.3 Software API

**NOTE** The algorithms described in this section apply to legacy designs (11n and 11ac Wave 1 Solutions). 11ac Wave 2 and later solutions do not use the below algorithms since the latest CTL table design captures all required power values. Qualcomm provides a scripting tool to assist in generation of the CTL file.

- *ar9300EepromSetTransmitPower*: This function invokes the target power calculation algorithm for the given regulatory domain and channel, sets up the power array registers with the per-rate target power table for CDD/Direct map mode of operation, and kicks off calibration. By default per-packet TPC is enabled. In this mode, this function also invokes the algorithms to compute per-rate target power tables for the CDD, TxBF, and STBC transmit modes.
- *ar9300EepromSetPowerPerRateTable*: Algorithm to compute the per-rate target power, factoring in the regulatory limits, antenna gain, target power tables, CTL, and multi chain factor.
- *ar9300InitRateTxPower*: Function to invoke the best per-rate target power calculation algorithm based on the mode of operation (11NG, 11NA, and so on).

The following functions populate the *txPower* table for the CCK, OFDM, and HT rates.

- *ar9300InitRateTxPowerCCK*
- *ar9300InitRateTxPowerOFDM*
- *ar9300InitRateTxPowerHT*

These functions compute per-rate TPC tables for TxBF, STBC, and CDD.

- *ar9300InitRateTxPowerTxBF*
- *ar9300InitRateTxPowerSTBC*
- *ar9300AdjustRateTxPowerCDD*

*ar9300GetPerRateTxBFFlag:* This function adjusts the TxBF flag (ON/OFF) based on the CDD power level for the rate for the MKK and ETSI domains. If the CDD power is constrained by the upper regulatory limit, then TxBF is disabled to avoid a full  $10 \cdot \log_{10} (N)$  back off.

*ar9300DumpRateTxPower:* This function dumps the per-rate power table for TxBF, CDD, and STBC modes. For ETSI and MKK domains it also dumps the TxBF flags for each rate (that is, the flag computed based on the CDD TPC).

### 8.3.2.4 Regulatory limits

#### FCC

|                                   | ISM     | UNII-I         | UNII-II       | UNII-II EXT     | UNII-III        |
|-----------------------------------|---------|----------------|---------------|-----------------|-----------------|
| Band                              | 2.4 GHz | (5.1- 5.2) GHz | (5.2-5.3) GHz | (5.4-5.725) GHz | (5.725-5.8) GHz |
| Total Conducted Power Limit (dBm) | 30      | 17             | 24            | 24              | 30              |
| Antenna Gain Limit (dBi)          | 6       | 6              | 6             | 6               | 6               |

#### ETSI

|                                   | ISM     | UNII-I         | UNII-II       | UNII-II EXT     |
|-----------------------------------|---------|----------------|---------------|-----------------|
| Band                              | 2.4 GHz | (5.1- 5.2) GHz | (5.2-5.3) GHz | (5.4-5.725) GHz |
| Total Conducted Power Limit (dBm) | 20      | 23             | 23            | 30              |

#### MKK

|                                   | ISM     | UNII-I         | UNII-II       | UNII-II EXT     |
|-----------------------------------|---------|----------------|---------------|-----------------|
| Band                              | 2.4 GHz | (5.1- 5.2) GHz | (5.2-5.3) GHz | (5.4-5.725) GHz |
| Total Conducted Power Limit (dBm) | 23      | 23             | 23            | 23              |

#### Array Gain Tables

|                 | Array Gain recommended with FCC rules (dB) | Array Gain recommended with ETSI rules (dB) |
|-----------------|--|---|
| 1-Tx / 1-stream | 0  | 0   |

| 2-Tx / 1-stream | Array Gain recommended with FCC rules (dB) | Array Gain recommended with ETSI rules (dB) |
|-----------------|--|---|
| CDD             | 3  | 0   |
| STBC            | 0  | 0   |
| TxBF            | 3  | 3   |

| <b>2-Tx / 2-stream</b> | <b>Array Gain recommended with FCC rules (dB)</b> | <b>Array Gain recommended with ETSI rules (dB)</b> |
|------------------------|---|--|
| Direct Map             | 0   | 0  |
| TxBF                   | 3   | 3  |

| <b>3-Tx / 1-stream</b> | <b>Array Gain recommended with FCC rules (dB)</b> | <b>Array Gain recommended with ETSI rules (dB)</b> |
|------------------------|---|--|
| CDD                    | 4.8   | 0  |
| STBC + Walsh           | 4.8   | 0  |
| TxBF                   | 4.8   | 4.8  |

| <b>3-Tx / 2-stream</b> | <b>Array Gain recommended with FCC rules (dB)</b> | <b>Array Gain recommended with ETSI rules (dB)</b> |
|------------------------|---|--|
| CDD + Walsh            | 4.8   | 0  |
| TxBF                   | 4.8   | 4.8  |

| <b>3-Tx / 3-stream</b> | <b>Array Gain recommended with FCC rules (dB)</b> | <b>Array Gain recommended with ETSI rules (dB)</b> |
|------------------------|---|--|
| Direct Map             | 0   | 0  |
| TxBF                   | 4.8   | 4.8  |

### 8.3.2.5 Troubleshooting

1. HAL\_PWR\_MGMT\_DBG can be enabled to see the TPC values calculated for each rate at the time of channel change. For ETSI and MKK the TxBF Disable Flag table is also displayed with this debug enabled.  

```
Root> iwpriv wifi0 HALDbg 0x00200000
```
2. Packet logs can be used to validate the per-packet TPC value. The power level to be used with each of the rate series is specified in the descriptor.
3. iwconfig <interface> will display the TxPower mapped as follows
  - Legacy mode = TxPower for 6Mbps
  - HT20 mode = TxPower for MCS0

The TPC values are computed for the txchainmask configured.

### 8.3.2.6 References

1. FCC Emissions Testing of Transmitters with Multiple Outputs in the same band (662911 D01  
Multiple Transmitter Output v01)
2. AR93xx/AR94xx/AR95xx EEPROM Device Configuration Guide
3. AR93xx Data Sheets

### 8.3.3 Additional feature

To mitigate interference, the added functionality allows transmit power configuration on a per management frame basis. Once the AP starts up and decides on the best transmit power to use in a particular environment, it can dynamically configure the transmit power for beacon, probe response, (re-) association request, (re-) association response, auth, disassociation and de-auth frames, without affecting present connections, i.e., without restart. The ATH\_PARAM\_TXPOW\_MGMT ioctl is used for this purpose in the DA path and OL\_ATH\_PARAM\_TXPOW\_MGMT ioctl is used in the partial offload path.

The commands used to do so are as follows:

1. To configure transmit power for a particular frame (offload and DA both):  

```
iwpriv wifiN set_txpow_mgmt <frame-subtype> <desired-transmit-power-for-the-frame>
```
2. To obtain transmit power configured for a particular frame with subtype value ‘frame-subtype’ (offload and DA both):  

```
iwpriv wifiN get_txpow_mgmt <frame-subtype>
```

The following are the frame subtype value in the command with standard IEEE conventions:

- Association Request - 0x00
- Association Response - 0x10
- Re-Association Request - 0x20

- Re-Association Response - 0x30
- Probe Response - 0x50
- Beacon - 0x80
- Disassoc - 0xa0
- Auth - 0xb0
- De-auth - 0xc0

To undo the effects of tx power set for a particular management frame type, use the following command:

```
iwpriv wifiN set_txpow_mgmt <frame-subtype> 255
```

The power value is configured in the HTT descriptor for frames sent in the offload path and in the tx descriptor for frames sent in the direct attach path. This feature is not valid for hardware generated frames.

## 8.4 Configure transmit power for management frames per SSID

Transmit power can be configured for beacon, probe responses, (re-) association request, (re-) association response, auth, disassociation and de-auth frames per ssid. This feature is applicable for both direct attach and partial offload radios. The power value configured for a particular frame can be altered dynamically without affecting present connections with clients (no restart is required).

The power configured for a management frame will be an 8 bit value. The frame will ultimately be transmitted at the least value out of – the user configured value, hardware capability, conformance test limits (CTL) and regulatory power limitations. The power provided by the user will be in dBm and will be used as a per-chain power such as the target-power/CTL. If the txpower is configured for both the radio and the SSID, then the txpower of the SSID overrides the txpower of the radio.

The command to set transmit power is designed specifically to exclude probe request frames and action frames as those can be hardware generated as well and as we have no provision to set transmit power for hardware generated frames for this feature, it is undesired to have a mismatch of transmit power of probe request frames generated by the same access point, depending on whether they are hardware generated or driver initiated. All hardware generated frames will be unaffected by any transmit power configuration done by the user.

### Partial offload

For partial offload mode, the tx power is set in the HTT descriptor for all management frames except the beacon and for the beacon it is set via a WMI command for each ol\_txrx\_vdev\_t structure on that radio. We use the OL\_ATH\_PARAM\_TXPOW\_MGMT ioctl to set as well as get the txpower of a particular management frame type.

The definition of the ioctl IEEE80211\_PARAM\_TXPOW\_MGMT in ieee80211\_ucfg\_setparam will accept 2 arguments instead of its usual one argument. The tx power values are stored in an array txpow\_mgt\_frm in the ol\_txrx\_vdev\_t structure. These values are then used in the htt\_h2t\_mgmt\_tx function to set tx power in the HTT descriptor of management frames.

Only if the frame type is beacon, instead of going to the ol\_pflow\_update\_vdev\_params function, it is re-directed to a separate function called wmi\_txpower\_vap\_beacon that sends a WMI command for the vdev to set the transmit power. This is done via the vdev parameter WMI\_VDEV\_PARAM\_MGMT\_TX\_POWER. If the WMI command is successful the transmit power value is additionally stored in the txpow\_mgt\_frm array so it can be queried by the user.

The definition of ioctl IEEE80211\_PARAM\_TXPOW\_MGMT in ieee80211\_ucfg\_getparam accept one argument as input, i.e., the frame subtype, in order to obtain the corresponding tx power set for that frame on the particular vap. This will access the array values of the txpow\_mgt\_frm array and the value is returned to the user.

### Direct Attach

For DA, the transmit power values configured by the user are stored in the av\_txpow\_mgmt\_frm array stored in the ath\_vap structure. For the purpose of setting and getting tx power values, we use the IEEE80211\_PARAM\_TXPOW\_MGMT ioctl.

The functionality to insert values in the av\_txpow\_mgmt\_frm array was placed in the ieee80211\_ucfg\_setparam and the functionality to query values stored in the array was placed in the ieee80211\_ucfg\_getparam function. These values were then programmed into the tx\_power\_cap variable stored in the HAL\_11N\_RATE\_SERIES structure, depending on the frame type for which the HAL\_11N\_RATE\_SERIES structure was being filled up.

- To configure transmit power for a particular frame (for both offload and DA):
 

```
iwpriv athN s_txpow_mgmt <frame-subtype> <desired-transmit-power-for-the-frame>
```
- To obtain transmit power configured for a particular frame with subtype value ‘frame-subtype’ (for both offload and DA):
 

```
iwpriv athN g_txpow_mgmt <frame-subtype>
```

The frame subtype value in the command complies with the standard IEEE conventions as follows:

- Association Request - 0x00
- Association Response - 0x10
- Re-Association Request - 0x20
- Re-Association Response - 0x30
- Probe Response - 0x50
- Beacon - 0x80
- Disassociation - 0xa0
- Authentication - 0xb0
- De-authentication - 0xc0

To undo the effects of Tx power set for a particular management frame type, use the following command:

```
iwpriv athN set_txpow_mgmt <frame-subtype> 255
```

## Sample scenarios of Tx power per SSID configuration

Consider a sample scenario with AP1- an access point with direct attach cards and AP2- an access point with partial offload cards. Also, a sniffer capable of sniffing frames on both 2.4G and 5G bands is used, connected to a system with Wireshark software installed.

After bringing up Wi-Fi with 2 VAPs on AP1, set to operate on channel X and sniffer configured to detect packets on channel X, we took a few sample captures of beacon frames, probe response frames and association response frames generated by AP1 on both the VAPs. This is performed to compare SSI signal of these frames generated by AP1, before and after transmit power configuration of those frame types. (As we did not have a power-meter at our disposal, the unit tests only check if setting the transmit power of a frame made any difference to its SSI signal. Because the user is setting absolute transmit power per chain, the final test procedure should be to measure absolute power of the frames generated by the AP).

The SSI signal of some of these frames were noted down from the frame capture to form a rough range of values the SSI signal can take.

The following command is used to set transmit power for beacon frames,

```
iwpriv athN s_txpow_mgmt <frame-subtype> <desired-transmit-power-for-the-frame>
```

probe response frames and association response frames are as follows:

```
iwpriv ath0 s_txpow_mgmt 0x80 15
iwpriv ath01 s_txpow_mgmt 0x50 10
iwpriv ath0 s_txpow_mgmt 0x10 8.
```

The following use cases illustrate the sniffer capture of these frames with the following command configured:

```
iwpriv athN s_txpow_mgmt <frame-subtype> <desired-transmit-power-for-the-frame>
```

1. Create 2 VAPs one on 2.4 GHz and one on 5GHz. Set the txpower for each VAP separately for any one management frame, say Association response. Using the sniffer sniff for the Association response frame and check the txpower for each VAP. It is seen that the txpower is set in accordance with the minimum of the user configured value, hardware capability, conformance test limits (CTL) and regulatory power and it may be different for both the VAPs if the user configured value takes effect.
2. Create 2 VAPs both on 2.4 GHz. Set the txpower for each VAP separately for any one management frame, say probe response. Using the sniffer sniff for the probe response frame and check the txpower for each VAP. It is seen that the txpower is set in accordance with the minimum of the user configured value, hardware capability, conformance test limits (CTL) and regulatory power and it may be different for both the VAPs if the user configured value takes effect.
3. Create 2 VAPs both on 5 GHz. Set the txpower for each VAP separately for any one management frame, say Authentication response. Using the sniffer sniff for the Authentication response frame and check the txpower for each VAP. It is seen that the txpower is set in accordance with the minimum of the user configured value, hardware capability, conformance test limits (CTL) and regulatory power and it may be different for both the VAPs if the user configured value takes effect.

4. Create 2 VAPs one on 2.4 GHz and one on 5 GHz. Set the txpower for each radio separately for any one management frame, say Association response. Using the sniffer sniff for the Association response frame and check the txpower for each radio. Now set the txpower for each VAP separately for any one management frame, say Association response. It is seen that the txpower is set in accordance with the minimum of the user configured value, hardware capability, conformance test limits (CTL) and regulatory power and it may be different for both the VAPs if the user configured value takes effect.
5. Create 2 VAPs one on 2.4 GHz and one on 5 GHz. Set the txpower for each radio separately for any one management frame, say Association response. Using the sniffer sniff for the Association response frame and check the txpower for each radio. Now set the txpower for each VAP separately for any one management frame, say Association response. It is seen that the txpower for both the VAPs is set in accordance to the values set for the VAPs. Now again when the values for VAPs is set to 255 then the values set for the radio will be taken into account.
6. Create 2 VAPs one on 2.4 GHz and one on 5 GHz. Set the txpower for each radio separately for any one management frame, say Association response. Using the sniffer sniff for the Association response frame and check the txpower for each radio. Now set the txpower for each VAP separately for any one management frame, say Association response. It is seen that the txpower is set in accordance with the minimum of the user configured value, hardware capability, conformance test limits (CTL) and regulatory power and it may be different for both the VAPs if the user configured value takes effect. Now again when the values for VAPs is set to 255 then the values set for the radio will be taken into account if that is the least. Now when the radio's value is set to 255 then the original txpower will be set.
7. Create 2 VAPs one on 2.4 GHz and one on 5 GHz. Set the txpower for 2.4 GHz radio for any one management frame, say Association response. Using the sniffer sniff for the Association response frame and check the txpower for each radio. Now set the txpower for each VAP separately for any one management frame, say Association response. It is seen that the txpower is set in accordance with the minimum of the user configured value, hardware capability, conformance test limits (CTL) and regulatory power and it may be different for both the VAPs if the user configured value takes effect. Now again when the values for VAPs is set to 255 then the values set for the 2.4 GHz radio will be taken into account for that particular VAP if that is the least and for the other VAP the original power value is retained. Now when the radio's value is set to 255 then the original txpower will be set for 2.4 GHz VAP as well if that is the least.
8. Create 2 VAPs both on 5 GHz. Set the txpower for the radio for any one management frame, say Association response. Using the sniffer sniff for the Association response frame and check the txpower. Now set the txpower for each VAP separately for any one management frame, say Association response. It is seen that the txpower is set in accordance with the minimum of the user configured value, hardware capability, conformance test limits (CTL) and regulatory power and it may be different for both the VAPs if the user configured value takes effect. Now again when the values for VAPs is set to 255 then the values set for the radio will be taken into account if it's the least. Now when the radio's value is set to 255 then the original txpower will be set.

## 8.5 Advanced Enterprise for 10.4

This section lists all Advance Enterprise features already implemented in 10.4, along with detailed information about each feature.

### 8.5.1 Advanced Enterprise API Support

This section lists APIs currently supported for Advanced Enterprise in direct attach mode.

#### 8.5.1.1 Get rate-power table

Given the current operation mode and channel, Qualcomm Technologies provides an API for UMAC to query the current rate-power table information maintained by target firmware (FW). UMAC expects a pointer pointing to the actual rate-power table data structure, which could be defined as the following:

```
struct rate_power_tbl {
    u_int8_t      rateIdx;          /* rate index in the rate table */
    u_int32_t     rateKbps;         /* transfer rate in kbs */
    u_int8_t      rateCode;         /* rate for h/w descriptors */
    u_int8_t      txbf: 1,           /* txbf eligible */
                  stbc: 1,           /* stbc eligible */
                  chain1: 1,          /* one-chain eligible */
                  chain2: 1,          /* two-chain eligible */
                  chain3: 1;          /* three-chain eligible */
    int16_t       txpower[AR9300_MAX_CHAINS];   /* txpower for different
chainmasks, in step of 0.5 dBm */
#ifndef ATH_SUPPORT_TxBF
    int16_t       txpower_txbf[AR9300_MAX_CHAINS];
#endif
    int16_t       txpower_stbc[AR9300_MAX_CHAINS];
};
```

For each available transmission rate, it shows whether it supports TxBF/STBC/1-chain/2-chain/3-chain. In case of TxBF or STBC, it also shows the Tx power limit in txpower\_txbf[] and txpower\_stbc[] respectively.

The related WMI command is WMI\_PDEV\_GET\_TPC\_CONFIG\_CMDID.

#### 8.5.1.2 Get maximum and minimum TX power

During the initialization phase, UMAC expects a list of available channels based on the current operation mode and country code (or regulatory domain code). In addition of the channel list, UMAC also needs the max and min TX power on each channel in the list. This is the max and min transmission power for all transmission rates. With this information, UMAC can determine the correct range of transmission power and feed into controller side (if any).

Currently, the channel list information along with the max/min TX power is provided to UMAC via the (\*setup\_channel\_list)(...) API.

### 8.5.1.3 Set TX power limit

The API to specify user level TX power limit is provided. The target firmware will take this user-level TX power limit into considerations, along with board-level power limit and CTL power limit.

The related WMI commands are WMI\_PDEV\_PARAM\_TXPOWER\_LIMIT2G and WMI\_PDEV\_PARAM\_TXPOWER\_LIMIT5G.

### 8.5.1.4 Set maximum rate per client

The API to specify the maximum allowed transmission rate for a client is provided.

- The user-specified max transmission rate should be among the negotiated rate set between the AP and client.
- Also the target firmware rate adaptation scheme should take this as one of the inputs to determine the actual transmission rate for frames destined to the specified client.
- This is the upper bound of transmission rate instead of fixed rate.

The related WMI command is WMI\_PEER\_ASSOC\_CMDID.

### 8.5.1.5 Set user-defined antenna gain

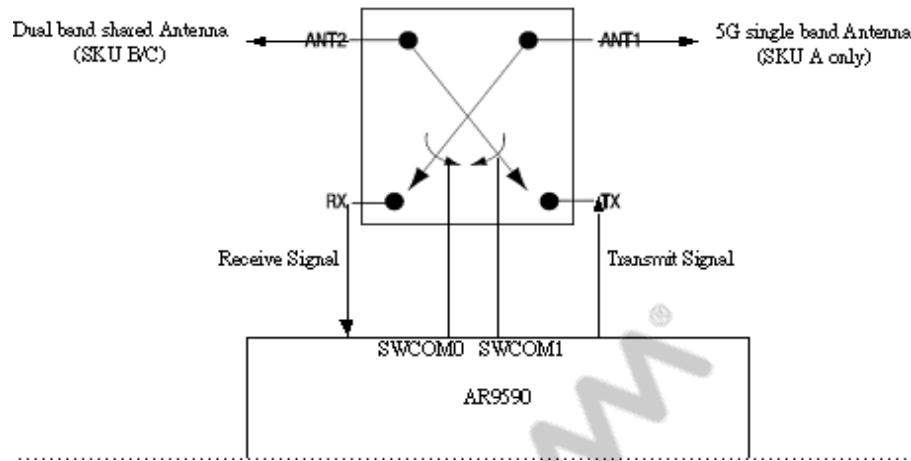
**NOTE** The algorithms described in this section apply to legacy designs (11n and 11ac Wave 1 Solutions). 11ac Wave 2 and later solutions do not use the below algorithms since the latest CTL table design captures all required power values. Qualcomm provides a scripting tool to assist in generation of the CTL file. The API to set the antenna gain value in addition to the default antenna gain value stored in EEPROM is provided. The antenna gain value is set for 2G and 5G radio separately. The TX power calculation in target firmware takes the following as inputs to determine the power limit without going beyond the regulatory power limit:

```
MIN( MAX(user defined antenna gain, antenna gain from EEPROM), max regulatory antenna gain);
```

The related WMI command is WMI\_PDEV\_PARAM\_ANTENNA\_GAIN

### 8.5.1.6 Set dynamic antenna selection

Some Qualcomm Technologies chipsets have an RF switch that provides the option to dynamically select the antenna for the radio. For example, the external antenna 5 GHz radio can either use uni-band 5 GHz antenna or share the same dual-band antenna with 2.4 GHz radio. The antenna selection is via GPIO-like pin control SWCOM, as shown in the figure below with AR9590 as an example:



At load time, default SWCOM values will be set based on EEPROM values. During runtime, an API to adjust the RF switch state and select antenna combinations is provided. The related WMI command is `WMI_PDEV_SET_ANTENNA_SWITCH_TABLE_CMDID`.

#### 8.5.1.7 Set CABQ ready time

An API to adjust the CAB queue ready time is provided. CAB queue is used to deliver multicast frames, and the trigger of such delivery is happened immediately after beacon transmission. The CABQ ready time specifies the maximum time for CABQ transmission within each beacon interval.

The value passed should be the percentage of beacon interval. For example, passing CABQ ready time of 80 causes at most 80% of beacon interval time to be used for CABQ.

This configuration should be done during TXQ setup.

The related WMI command is `WMI_VDEV_PARAM_CABQ_MAXDUR`.

#### 8.5.1.8 Set/get RX filter

The API to set/get the user-specified RX filter is provided. Promiscuous mode should be supported with this API.

Currently, the target controls the RX filter in QCA980x/QCA989x and QCA999x/998x.

The related WMI command is `WMI_PDEV_PARAM_RX_FILTER`.

#### 8.5.1.9 Set per-client TX chainmask

In addition to the global TX chainmask setting, another API to control the TX chainmask on per client base is provided. This is helpful with legacy client with single antenna, which usually has issue of receiving frames delivered with CDD enabled.

The API is supposed to specify the TX chainmask for all data and management frames destined to a single client. It might not change the TX chainmask setting for beacon frames as well as HW self-generated frames such as RTS, CTS and ACK.

The related WMI command is `WMI_PEER_NSS`.

#### **8.5.1.10 Set user-defined CTL table**

CTL tables are usually hard-coded in EEPROM for each device. Currently, only FCC is supported, Japan and ETSI in QCA CTL table structure. Advance Enterprise has more combinations, which makes it hard to store everything in EEPROM and confront with Qualcomm Technologies CTL format. Instead, the CTL API which accepts the QCA compatible CTL format from customers is provided. All the conversions of CTL format and domain mapping are supposed to be done by customer itself.

At load time, QCA driver will load the default CTL table from EEPROM. Before bringing up the WLAN device, this API must be called to replace the default CTL table with the new table specified by user. The API will copy the new CTL table into WLAN driver and perform an internal reset, in order to recalculate the power table, which will be exposed to upper layer.

The related WMI command is `WMI_PDEV_SET_CTL_TABLE_CMDID`.

#### **8.5.1.11 Set regdomain without country code**

The API to set the regulatory domain is provided, regardless of the country code specified. The API must pop up the correct channel list, recalculate the rate-power table, and expose this information to UMAC.

The related WMI command is `WMI_PDEV_SET_REGDOMAIN_CMDID`.

#### **8.5.1.12 RAW Mode Support**

The ability to send MPDUs in raw mode will be supported. In this case, SW/FW/HW should not perform encapsulation on the frame. Currently the raw frame (either data or management) is sent through WMI interface without any aggregation. Moreover, the raw frame will be queued into non-paused TID queue for Tx scheduling (refer to `OFFCHAN_EXT_TID_NONPAUSE`).

The related WMI commands are `WMI_SCAN_OFFCHAN_DATA_TX` and `WMI_SCAN_OFFCHAN_MGMT_TX`.

#### **8.5.1.13 Off Channel Operation**

Advance Enterprise AP goes off the channel frequently to either send out management/data frames or listen on the channel. To accomplish this requirement, the following actions are performed in SW/FW:

- Pause all client nodes and the VAP node
- Stop the Tx DMA logic
- Flush out frames in HW queue. The following two possibilities can arise:

- Frames have been transmitted out by HW. In this case, the TX status must already be updated. These frames should go through normal Tx complete process;
- Frames not yet transmitted by HW. In this case, they are removed from HW queue and re-queue back in the per-client TID queue for future transmission.
- Switch to off-channel. There must be no beacon transmission during the off-channel period.
- Restart the Tx and Rx logic.
- During off channel, a mechanism should be provided to send out frame even when all nodes are in paused state.
- Switch back to home channel. Reenable beacon transmission.
- Unpause all client nodes as well as VAP node

Currently, the off-channel operation is implemented with the FW scan logic. The related WMI command is WMI\_START\_SCAN\_CMDID. An iwpriv interface to test this off-channel operation namely, `iwpriv athX offchan_tx_test <ieee channel>`, is provided.

#### **8.5.1.14 Extended 5.825-5.875 GHz Indian Band**

A new regdomain code is added with newly added channel set to achieve this requirement.

#### **8.5.1.15 DFS Related**

This section lists Advance Enterprise requirements in DFS area.

##### **Enable/disable DFS on channel change**

When performing channel change, an option to disable the DFS (radar detection) even if the channel is a DFS channel is provided.

The related DFS flag is `ignore_dfs`.

##### **Enable/disable CAC on channel change**

CAC (Channel Availability Check) time is the fixed interval during which a system shall monitor a channel for the presence of radar prior to initiating a communication link on that channel. The option to disable the CAC waiting time during channel change is provided.

The related DFS flag is `ignore_cac`.

#### **8.5.1.16 Throttling Tx/Rx Processing**

##### **Tx loop limit**

Limit the number of frames pushed into HWQ per TX scheduling. This should apply to both non-aggregate and aggregate frames.

## Rx loop limit

The Rx loop limit is required for Rx processing because IOS runs in a single context and it is up to the tasks to relinquish the CPU for other tasks. For IOS, if a process is running and a WLAN interrupt occurs, the ISR routine is called to process the interrupts and returns. Then the tasklet routine is called (presumed to be called in interrupt context as well) and runs until no Rx frames remain in the Rx queue. Normally, it does not yield to other tasks until the Rx queue becomes empty or on any error condition occurs. Note that Rx processing should be limited to only  $N$  frames at a time.

The Rx loop limit feature allows limiting the number of frames to user configured value and defers the processing of the remaining Rx frames after a certain amount of time, thus avoiding the scenario where Rx processing takes up resources so other processes have no chance to run.

## 8.6 Thermal Mitigation

This section defines the interface between Thermal Mitigation Daemon (TMD), WLAN host driver, and WLAN firmware, various configuration options, and working of the thermal mitigation (Throttling) scheme.

### 8.6.1 Objectives

The QCA9980 is a 4x4 chip targeted for enterprise class products giving a throughput of Gigabits per second. In CPU-intensive scenarios, there are chances of QCA9980 chip's temperature shooting up to critical levels. Thermal mitigation is implemented to avoid complete breakdown of the chip.

### 8.6.2 Interfaces

#### 8.6.2.1 WLAN Driver and TMD

TMD is an user space application that changes the behavior of thermal mitigation by configuring various parameters. TMD reads various parameters from a configuration file and chip temperature from /sysfs entry created by the WLAN driver, to export chip temperature to the user space. For details on TMD, see [Section 8.6.6](#).

The WLAN driver creates five /sysfs entries at attach time:

- `/sys/class/net/wifiN/thermal/mode` [permission: RW, possible values: “enabled” and “disabled”]
- `/sys/class/net/wifiN/thermal/temp` [permission: R, possible values: Int]
- `/sys/class/net/wifiN/thermal/thlvl` [permission: RW, possible values: 0, 1, 2, 3 (more levels may be added in the future)]
- `/sys/class/net/wifiN/thermal/dc` [permission: RW, possible values (milliseconds): +ve int ]
- `/sys/class/net/wifiN/thermal/off` [permission: RW, possible values (off percent): [0, 100] ]

|           |   |
|-----------|---|
| Mode      | Set as “enabled” to enable thermal mitigation and “disabled” to disable thermal mitigation. The “enabled” and “disabled” options are mapped to 1 and 0 in the WLAN driver respectively.   |
| Temp      | Read-only entry for reading sensor temperature reported by the firmware to host.  |
| Thlvl/Off | Entries for setting the off percent for a specific thermal zone. These can be set in any order. Setting both of these correspond to 1 configuration. If a read operation is issued on these entries, it returns 0 or the last value set depending on whether the command is pending or completed. |
| Dc        | It affects the duty cycle of the specific radio (duty cycle of all thermal zones/levels).   |

### 8.6.3 WLAN Driver and Firmware

There is only one WMI command WMI\_TT\_SET\_CONF\_CMDID, implemented to configure the firmware with thermal mitigation parameters. The host also subscribes to the WMI event WMI\_TT\_STATS\_EVENTID to receive the current chip temperature and other stats regarding thermal mitigation.

The WMI command WMI\_TT\_SET\_CONF\_CMDID has the following configuration structure:

```
#define TT_LEVELS 4 /* Number of thermal throttle zones */
typedef struct {
    A_UINT32 tmplwm; /* Sensor value in Celsius at which the system should
exit to lower thermal zone/state */
    A_UINT32 tmphwm; /* Sensor value in celsius at which the system should
exit to higher state */
    A_UINT32 dcoffpercent; /* Duty cycle off percent 0-100. 0 means no
off, 100 means no on (shutdown the phy). */
    A_UINT32 prio; /* Disable only the queues that have lower priority
than configured. 0 means disable all queues */
} tt_level_config_t;

typedef struct {
    A_UINT32 enable; /* 0:disable, 1:enable */
    A_UINT32 dc; /* Duty cycle in ms */
    A_UINT32 dc_per_event; /* How often (for how many duty cycles) the
firmware sends stats to host */
    tt_level_config_t levelconf[TT_LEVELS]; /* Per zone config parameters
*/
} tt_config_t
```

The WMI event is expected to have the following fields in payload:

```
typedef struct {
    A_UINT32 levelcount; /* count of each time TT entered this state */
    A_UINT32 dccount; /* total number of duty cycles spent in this
state. * this number increments by one each time we are in this state
* and we finish one full duty cycle.
*/
} tt_level_stats_t;

typedef struct {
    tt_level_stats_t levstats[TT_LEVELS]; /* stats for each level */
```

```

    A_UINT32 temp; /* temperature reading in Celsius */
    A_UINT32 level; /* current level */
} tt_stats_t;

```

## 8.6.4 Thermal tool

The thermal tool is an user space tool implemented to configure the various thermal mitigation parameters. Following options can be used for configuration:

- set specifies set operation
- get specifies get operation. Reads config from driver and displays on screen.
- i interface name wifi0 or wifi1
- e 1: enable, 0: disable
- et event time in duty cycle units [for example, 10 means for each 10 duty cycles, firware will send 1 event]
- dc duty cycle in milliseconds
- dl bitmap of 4 log levels. By default, only log level 1 (only error messages) is enabled.
- pN thermal policy for level/zone N [only policy Queue Pause:1 is supported as of now]
- loN low threshold for level N
- hiN high threshold for level N
- offN Tx Off percentage for level N
- qpN disables all Tx queues having priority less than the configured value for level N

**NOTE** Option “-set” should be used to set the configuration; option “-get” should be used to read the configuration. Help string is displayed whenever there is a mistake while typing the command.

Example:

- Set operation should be like #thermaltool -i wifiN -set -e 1 -dc XXX ...
- Get operation should be like #thermaltool -i wifiN -get

## 8.6.5 Implementation

### 8.6.5.1 WLAN Host

The WLAN host is responsible for exporting the current sensor temperature of a chip to the user space via a sysfs entry, receiving the configuration values from TMD or thermal tool, and configuring the firmware accordingly. The thermal tool can also query the current configuration and thermal stats from the driver.

The various sysfs entries exported by the driver are:

- **/sys/class/net/wifiN/thermal/mode** – this mode can have two possible values, “enabled” and “disabled”. When TMD sets the mode as “enabled”, thermal mitigation is enabled; when TMD sets the mode as “disabled”, thermal mitigation is disabled.
- **/sys/class/net/wifiN/thermal/temp** – this is a read-only sysfs which exports the current temperature of a chip to TMD. The temperature of the chip is informed to the WLAN host by the firmware, by sending a periodic event.
- **/sys/class/net/wifiN/thermal/thlvl** – this marks the current thermal level for which the TMD wants to modify the off percentage.
- **/sys/class/net/wifiN/thermal/off** – this marks the Tx off duration for level “thlvl”.
- **/sys/class/net/wifiN/thermal/dc** – this marks the duty cycle in milliseconds. Out of dc milliseconds, Tx will be off for  $(dc * off / 100)$  milliseconds and will be on for  $(dc * (100 - off) / 100)$  milliseconds.

At attach time, the WLAN host sends WMI\_TT\_SET\_CONF\_CMDID command to the firmware along with the default configuration, if the firmware advertises its capability to support thermal mitigation by setting the WMI\_SERVICE\_TT flag in the WMI service capability bitmap.

The WMI\_TT\_SET\_CONF\_CMDID command has tt\_config\_t as the payload.

See [Figure 8-3](#) for the flow Diagram.

### 8.6.5.2 WLAN Firmware

The firmware checks if it is capable of supporting thermal mitigation and notifies the host through the WMI service capability bitmap in the ready event. On receipt of the WMI command WMI\_TT\_SET\_CONF\_CMDID from the host, it stores the configuration options for future use. If thermal mitigation is enabled, it starts monitoring the temperature of thermal sensors, and keeps pausing or resuming the configured Tx queues for times calculated from dc and dc off percent of the current thermal level. In other words, the firmware starts a cycle of dc milliseconds; during this time interval, the Tx queues are active for  $(dc * (100 - off) / 100)$  ms and paused for  $(dc * off / 100)$  ms.

The firmware periodically (each dc\_per\_event \* dc milliseconds) reads the thermal sensor reading and other stats, puts them in struct tt\_stats\_t and sends it to the host as a payload of the WMI\_TT\_STATS\_EVENTID event.

So far, only the Tx queue pause policy is implemented for cooling the chip.

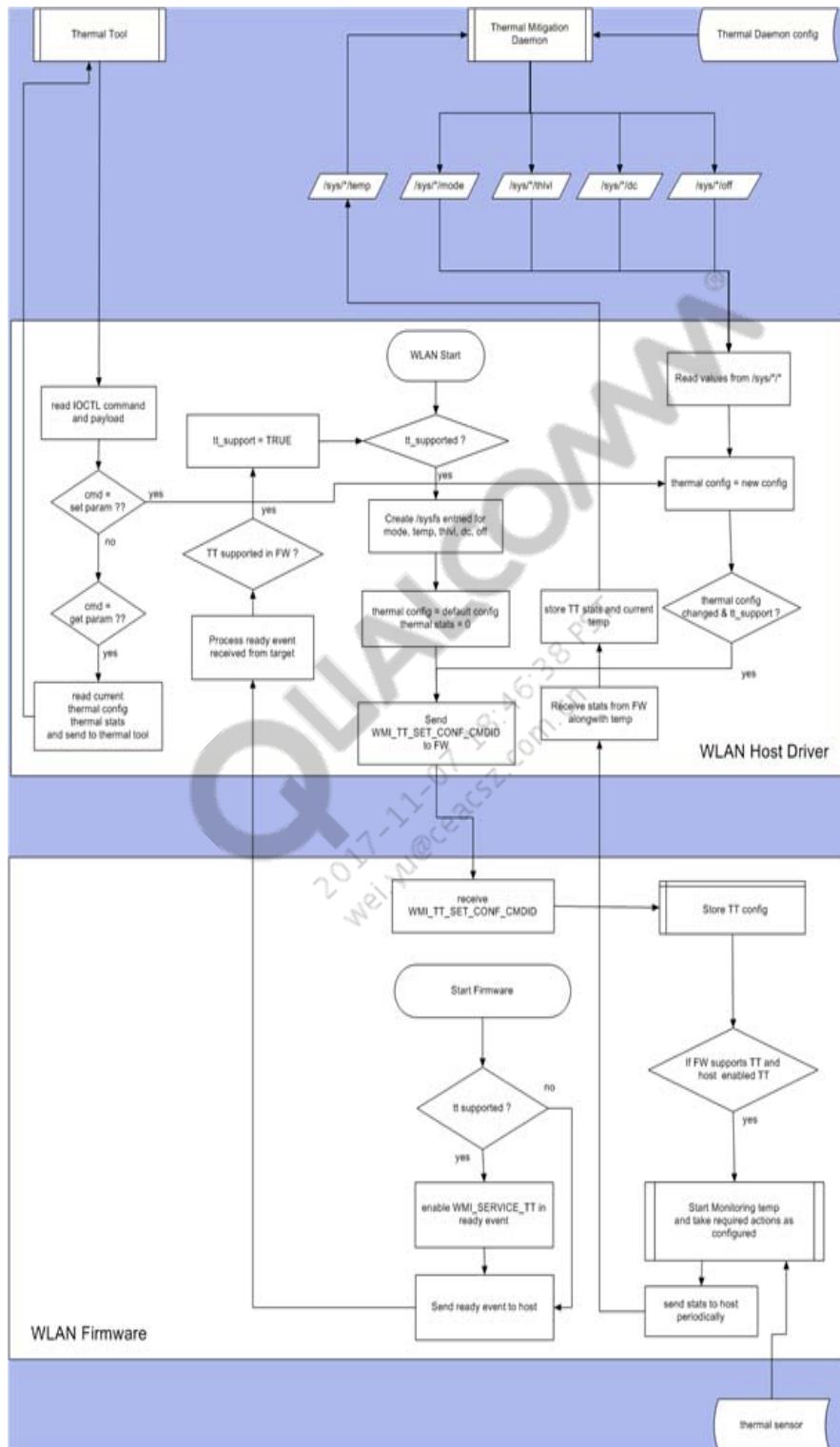


Figure 8-3 WLAN Firmware

## 8.6.6 TMD-WLAN Interface

### 8.6.6.1 Overview

Thermal mitigation is to be done by a dedicated daemon (TMD). The idea is to have the mitigation done at one centralized module which runs in the user space and can be configured for any number of thermal zones and modules, with the help of a configuration file. Since the daemon is running in the user space, the max CPU frequency setting, Wi-Fi throughput restriction, antenna power reduction etc., are done through sysfs entries. The temperature of various sensors are also read via sysfs entries.

### 8.6.6.2 Configuration File Format

```
[temperature_sensor_name]
sampling <Time in milliseconds>
thresholds t1 t2 t3 ... t(n)
thresholds_clr tc1 tc2 tc3 ... tc(n)
actions a1 a2 a3 ... a(n)
action_info ail ai2 ai3 ... ai(n)
```

For example, the entries for Krait sensors could look like:

```
[tsens_tz_sensor0]
sampling 1000
thresholds 70
thresholds_clr 67
actions none
action_info 0

[tsens_tz_sensor8]
sampling 1000
thresholds 70 90 95 100 105 110 115 120
thresholds_clr 67 85 90 95 100 105 110 115
actions cpu cpu cpu cpu cpu cpu cpu
shutdown
action_info 1728000 1188000 810000 648000 540000 487000 384000
5000
```

- **Temperature\_sensor\_name** – name/ID of the temperature sensor. For example, in Krait, it could be tsens\_tz\_sensor0 to tsen\_tz\_sensor10.
- **Sampling** – time duration between each sample in milliseconds.
- **Thresholds** – upper limit for the number of checkpoints for a particular sensor. There could be one or many threshold values. TMD takes corresponding actions when the temperature is within thresholds and thresholds\_clr.
- **Thresholds\_clr** – lower limit for the number of checkpoints for a particular sensor. Thresholds\_clr along with the thresholds forms the range for which the corresponding action applies.
- **Actions** – the action to be performed when the corresponding threshold limit is hit. For example, it could be cpu (for Krait-related action), Wlan (for Wi-Fi action) or shutdown.

- **Action\_info** – information on the action to be performed. For example, in Krait, the max frequency for the respective threshold range is specified. The increase/decrease in frequency plays a critical part in thermal mitigation.

The TMD is a straight forward program. On initialize, TMD reads the configuration from the configuration file. It spawns a thread for each temperature sensor entry. The threads poll the temperature sensor values based on the sampling rate. Based on the current temperature, action is taken. The threads are alive till TMD closes.

## 8.7 Modify Tx chain-mask without reconnecting clients

For enabling power consumption in the most optimal manner, a capability to apply tx chain-masks without STAs being disconnected/reconnected, when required, is introduced. This feature enables the user to configure the Tx chain masks on the fly in the AP board without disconnecting the associated clients. Rx chain mask should remain intact. Because dynamically changing Tx chain masks affects the MU functionality, it is necessary to disable the MU feature when this TX-chain masks feature is enabled and also the KPI for the throughput cannot be guaranteed.

Run ping or IxChariot traffic in single user (SU) mode and modify the Tx chain masks dynamically using the `iwpriv wifiX txchainsoft 0xN` command. Alternatively, run ping or IxChariot traffic in multiple user (MU) mode and modify the Tx chain masks dynamically using the `iwpriv wifiX txchainsoft 0xN` command.

### 8.7.1 Sample configuration scenario

The following is an example of the AP configuration:

```
config wifi-device wifi0
    option type qcawifi
    option channel auto
    option macaddr 00:a0:c6:01:21:38
    option hwmode 11ac
    # REMOVE THIS LINE TO ENABLE WIFI:
    option disabled 0

config wifi-iface
    option device wifi0
    option network lan
    option mode ap
    option ssid gan-2g
    option encryption none

config wifi-device wifil
    option type qcawifi
    option channel 36
    option macaddr 00:a0:c6:01:20:da
    option hwmode 11ng
    # REMOVE THIS LINE TO ENABLE WIFI:
    option disabled 1

config wifi-iface
    option device wifil
```

```

option network lan
option mode ap
option ssid gan-5g
option encryption none

```

The following is an example of STA setup in SU mode:

```

config wifi-device wifi0
    option type qcawifi
    option channel auto
    option macaddr 00:a0:c6:01:21:38
    option hwmode 11ac
    # REMOVE THIS LINE TO ENABLE WIFI:
    option disabled 0

config wifi-iface
    option device wifi0
    option network lan
    option mode sta
    option ssid gan-2g
    option encryption none
    option wds 1

config wifi-device wifi1
    option type qcawifi
    option channel 36
    option macaddr 00:a0:c6:01:20:da
    option hwmode 11ng
    # REMOVE THIS LINE TO ENABLE WIFI:
    option disabled 1

config wifi-iface
    option device wifi1
    option network lan
    option mode sta
    option ssid gan-5g
    option encryption none
    option wds 1

```

The following is an example of STA setup in MU mode:

```

config wifi-device wifi0
    option type qcawifi
    option channel auto
    option macaddr 00:a0:c6:01:21:38
    option hwmode 11ac
    # REMOVE THIS LINE TO ENABLE WIFI:
    option disabled 0

config wifi-iface
    option device wifi0
    option network lan
    option mode sta
    option ssid gan-2g
    option encryption none
    option wds 1

config wifi-device wifi1

```

```

option type      qcawifi
option channel   36
option macaddr   00:a0:c6:01:20:da
option hwmode    11ng
# REMOVE THIS LINE TO ENABLE WIFI:
option disabled 1

config wifi-iface
    option device    wifi1
    option network   lan
    option mode      sta
    option ssid      gan-5g
    option encryption none
    option wds      1

```

Run ping / Ixchariot traffic in the following sequence:

- iwpriv wifiX txchainsoft 1
- ping <ip address>

The chain mask can be changed ensuring that ping is not stopped, although ping traffic might be impacted slightly and clients continue to remain connected. Change the Tx chain mask in ascending order and then in descending order (or vice versa). Also, a user can change the chain mask randomly to assure its stability. These changes can be performed in both SU mode and MU mode.

Increasing or decreasing the Tx chain appropriately impacts the NSS count listed under Firmware stats 6 section of the txrx\_fw\_stats command. For example, with iwpriv wifiX txchainsoft 1 entered, the following output is observed:

```

root@OpenWrt:/# iwpriv ath1 txrx_fw_stats 6
root@OpenWrt:/# [27093.437687] TX Rate Info:
[27093.440664] MCS counts (0..9): 0, 0, 0, 13, 11, 17, 33, 33, 33
[27093.447102] MCS counts SU (0..9): 0, 0, 0, 13, 11, 17, 33, 33, 33
[27093.453467] MCS counts MU (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0
[27093.459180] SGI counts (0..9): 0, 0, 0, 0, 0, 0, 29, 26, 28
[27093.464974] NSS counts: 1x1 140, 2x2 0, 3x3 0 4x4 0
[27093.469926] BW counts: 20MHz 140, 40MHz 0, 80MHz 0, 160MHz 0
[27093.475684] Preamble (O C H V) counts: 0, 37, 0, 140
[27093.480498] STBC rate counts (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0
[27093.486579] LDPC Counts: 140
[27093.489424] RTS Counts: 1
[27093.492013] Ack RSSI: 26

```

With iwpriv wifiX txchainsoft 2 entered, the following output is observed:

```

root@OpenWrt:/# iwpriv ath1 txrx_fw_stats 6
root@OpenWrt:/# [27104.114079] TX Rate Info:
[27104.116439] MCS counts (0..9): 0, 0, 0, 13, 11, 17, 45, 41, 33
[27104.123037] MCS counts SU (0..9): 0, 0, 0, 13, 11, 17, 45, 41, 33
[27104.129099] MCS counts MU (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0
[27104.134910] SGI counts (0..9): 0, 0, 0, 0, 0, 0, 41, 30, 28
[27104.140743] NSS counts: 1x1 140, 2x2 20, 3x3 0 4x4 0
[27104.145690] BW counts: 20MHz 160, 40MHz 0, 80MHz 0, 160MHz 0
[27104.151322] Preamble (O C H V) counts: 0, 37, 0, 160
[27104.156261] STBC rate counts (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0
[27104.162420] LDPC Counts: 160

```

```
[27104.165177] RTS Counts: 3
[27104.167798] Ack RSSI: 26
```

Similarly, with iwpriv wifiX txchainsoft 3 entered, the following output is observed:

```
iwpriv wifiX txchainsoft 3
```

```
root@OpenWrt:/# iwpriv ath1 txrx_fw_stats 6
root@OpenWrt:/# [27207.979106] TX Rate Info:
[27207.981329] MCS counts (0..9): 0, 0, 0, 0, 13, 11, 38, 61, 115, 52
[27207.987778] MCS counts SU (0..9): 0, 0, 0, 0, 13, 11, 38, 61, 115, 52
[27207.994285] MCS counts MU (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
[27207.999993] SGI counts (0..9): 0, 0, 0, 0, 0, 0, 0, 53, 98, 31
[27208.005812] NSS counts: 1x1 140, 2x2 40, 3x3 60 4x4 0
[27208.010767] BW counts: 20MHz 290, 40MHz 0, 80MHz 0, 160MHz 0
[27208.016410] Preamble (O C H V) counts: 0, 44, 0, 290
[27208.021342] STBC rate counts (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
[27208.027425] LDPC Counts: 290
[27208.030260] RTS Counts: 3
[27208.032880] Ack RSSI: 28
```

### 8.7.2 Known limitations

- MU is disabled.
- KPI for throughput cannot be committed.
- Traffic might be disturbed while changing the Tx chain masks abruptly.

## 8.8 Disable buffering of multicast frames for OL and DA mode stations in power-save mode

A user space IOCTL in the form of an iwpriv command is introduced to enable and disable buffering of multicast frames at AP when stations are in power-save mode in both offload (OL) and direct attach (DA) modes. After the AP disables buffering of multicast frame, clients which goes into power-save mode does not receive these frames.

The user can enable/disable this feature at any instance of time. The current design enables the multicast buffering if any one STA moves into power save mode and continues to buffer the multicast packets until all the associated STAs come out of power save mode.

Certain rare conditions might arise in which this feature might not be effective with the current design model. The following are such rare scenarios:

- It is not possible to employ separate conditional statements to unpause the NON-QOS TID because it might affect the connection pause SM and will bypass some of the power-save modifications.
- When the number of power save STAs are more than one, disabling the multicast buffering becomes ineffective as the current design pauses the NON-QOS TID only when one STA is in power-save mode.

- Enabling and disabling this feature should gracefully handle the conditions of pausing and unpausing the NON-QOS TIDs, such that when this feature is disabled and still there are some STAs in power save mode multicast buffering should be triggered without any issues.

To address the aforementioned challenges and enable this mechanism to function effectively, a truth table is designed to address all the corner conditions. The truth table, which is a mathematical table of Boolean logical operators, for the CABQ feature is as follows:

| Cabq_disable | Number_sleeping_STAs | Is_non_qos_tid_paused | Result   |
|--------------|----------------------|-----------------------|--|
| 1            | 1                    | x                     | Multicast buffering is not enabled and NON-QOS TID is not paused.  |
| 1            | >1                   | 1                     | Multicast buffering will be disabled and NON-QOS TID will be unpause.  |
| 1            | >1                   | 0                     | Multicast buffering is already disabled and NON-QOS TID will be in unpause state. Do not allow SM to unpause it again.                 |
| 1            | 0                    | 1                     | Multicast buffering will be disabled and NON-QOS TID will be unpause.  |
| 1            | 0                    | 0                     | Multicast buffering is already disabled and NON-QOS TID will be in unpause state. Do not allow state machine (SM) to unpause it again. |
| 0            | 1                    | x                     | Multicast buffering is enabled and NON-QOS TID will be paused.   |
| 0            | >1                   | 1                     | Multicast buffering is already enabled and NON-QOS TID will be in pause state. Do not allow SM to pause it again.                      |
| 0            | >1                   | 0                     | Multicast buffering is enabled and NON-QOS TID will be paused  |
| 0            | 0                    | 1                     | Multicast buffering will be disabled and NON-QOS TID will be unpause   |
| 0            | 0                    | 0                     | Multicast buffering is already disabled and NON-QOS TID will be in unpause state. Do not allow SM to unpause it again.                 |

The mechanism to prevent multicast frames from being buffered when a STA enters the power-save mode is implemented. When a STA enters power save mode, multicast frame will be buffered to content after beacon queue (CABQ) and sent at DTIM. This feature implements an iwpriv command to enable or disable multicast buffering. As a result, when a STA is in power-save mode, the multicast frames are transmitted as they are normally sent. However, the power-save mode STA does not receive these frames, whereas other STAs receive them.

When a radio operates in OL mode, all the multicast buffer and CABQ process are in target. In such cases, the host needs to send only a WMI message to the target to enable this do-not-buffer functionality.

In the Wi-Fi driver, the `iwpriv ath0 cabq_disable 1` command is implemented to enable this feature to avoid buffering multicast frames when STA is in power-save mode for both OL and DA modes. For OL mode, this command triggers a WMI message to target to enable this do-not-buffer or disable-CABQ feature.

After enabling this feature, on the multicast client, it is noticed that certain packets are missing because the multicast packets are delivered instantaneously as they are received, regardless of whether any of the STAs are in power-save mode. Also, in the sniffer capture, it is observed that multicast packets are delivered at different beacon intervals, unlike the behavior in power save mode.

Alternatively, enter the `iwriv ath0 cabq_disable 0` command to disable this feature to avoid buffering multicast frames when STA is in power-save mode for both OL and DA modes.

## 8.9 Support for the same Tx power-out in dBm with antenna across all chains

On QCA9984 chipsets, the functionality to add the antenna gain values in the board data file (BDF) on a per-channel and per-chain basis to achieve the equal actual Tx power-out with antenna is supported. Before the implementation of this functionality, the SW-programmed Tx power at antenna connector for each chain and respective external antenna gain are added. If the antenna gain values are different for each chain, then the actual Tx power-out with antenna is different for each chain.

For example, assume that the SW-programmed Tx power at antenna connector is 17dBm for all the chains, whereas the actual antenna gain is different power each chain (3 dB for chain0, 4 dB for chain1, 5 dB for chain2, 6 dB for chain3). In this case, the actual Tx power-out with antenna is different for each chain (20 dBm for chain 0, 21 dBm for chain1, 22 dBm for chain2, 23 dBm for chain3), whereas it needs to be the same for each chain.

Starting with QCA\_Networking\_2017.SP5.0 CSU1, the SW-programmed Tx power at antenna connector is adjusted by considering the actual antenna gain. This adjustment is performed as part of calibration. As a result, the actual Tx power-out with antenna is the same across all the chains.

**Table 8-1 Tx power-out in proposed scenario**

| Channel - 5180                               | Proposed scenario (with Implementation of FR) |         |         |         |
|--|---|---------|---------|---------|
| Chain Number                                 | Chain_0                                       | Chain_1 | Chain_2 | Chain_3 |
| BDF Tx_Power (dBm)                           | 17  | 17      | 17      | 17      |
| SW programmed Tx Power @ Ant connector (dBm) | 17  | 16      | 15      | 14      |
| Actual Antenna gain (dB)                     | 3   | 4       | 5       | 6       |
| Actual Tx Power out with Antenna, EIRP (dBm) | 20  | 20      | 20      | 20      |

The preceding table illustrates that SW programmed Tx power at antenna connector is different for each chain (17 dBm for chain0, 16 dBm for chain1, 15 dBm for chain2, and 14 dBm for chain3). Therefore, the actual Tx power-out with antenna is equal (20 dBm for chain 0, 20 dBm for chain1, 20 dBm for chain2, and 20 dBm for chain3).

A new BDF parameter, “Actual\_Antenna\_gain”, is added to provide the antenna gain values on per channel and chain basis. In the EEPROM structure, a two-dimensional array of a total size of 32 bytes is added to obtain the actual antenna gain values from BDF and to perform corresponding operations. For example, in the Actual\_antenna\_gain [8][4], 8 denotes the number of calibration channels and 4 indicates the number of chains.

The actual antenna gain values are added to the measured power obtained in calibration with respect to the per-channel basis and per-chain basis.

$$\text{Measured power} = \text{Measured power} + (\text{Actual antenna gain of current chain} - \text{Minimum antenna gain of all chains}) \quad (1)$$

Updated measured power is programmed in gain lookup table (GLUT) and power lookup table (PLUT). As part of mission mode transmission, modified power values are processed through the transmit power control (TPC) logic and the transmit power is adjusted by considering the antenna gain. A BDF flag, enable\_perchain\_antennagain, is added to enable or disable this feature. The expected memory consumption is 50 bytes in SRAM.

The following three cases arise, in which the target power can be less than, equal to, or greater than the CTL power:

### Target Power < CTL Power

**Table 8-2 Target Power less than CTL Power**

|                  | Chain0 | Chain1 | Chain2 | Chain3 |
|------------------|--------|--------|--------|--------|
| Target Power     | 17     | 17     | 17     | 17     |
| CTL Power        | 20     | 20     | 20     | 20     |
| Antenna Gain     | 3      | 4      | 5      | 6      |
| New Target Power | 17     | 16     | 15     | 14     |
| Output Power     | 20     | 20     | 20     | 20     |

### Target Power = CTL Power

Table 8-3 Target Power equal to CTL Power

|                  | Chain0 | Chain1 | Chain2 | Chain3 |
|------------------|--------|--------|--------|--------|
| Target Power     | 17     | 17     | 17     | 17     |
| CTL Power        | 17     | 17     | 17     | 17     |
| Antenna Gain     | 3      | 4      | 5      | 6      |
| New Target Power | 17     | 16     | 15     | 14     |
| Output Power     | 20     | 20     | 20     | 20     |

### Target Power > CTL Power

Table 8-4 Target Power greater than CTL Power

|   |                  | Chain0 | Chain1 | Chain2 | Chain3 |
|---|------------------|--------|--------|--------|--------|
|   | Target Power     | 17     | 17     | 17     | 17     |
|   | CTL Power        | 10     | 10     | 10     | 10     |
|   | Antenna Gain     | 3      | 4      | 5      | 6      |
| Consider Minimum gain(3dB)                | New Target Power | 10     | 9      | 8      | 7      |
|   | Output Power     | 13     | 13     | 13     | 13     |
| Consider Maximum gain(6dB)                | New Target Power | 13     | 12     | 11     | 10     |
|   | Output Power     | 16     | 16     | 16     | 16     |
| By changing CTL values as per requirement | New Target Power | 17     | 16     | 15     | 14     |
|   | Output Power     | 20     | 20     | 20     | 20     |

## 8.10 Transmit power control per SSID for control frames

In addition to the support for transmit power control per SSID for management frames in both DA and offload modes that existed in previous releases, the functionality to configure transmit power control per SSID for control frames is introduced in the QCA\_Networking\_2017.SPF.5.0 release. Enter the `iwpriv athX s_txpow_ctl frame_subtype` command to specify the Tx power control for control frames. Enter the `iwpriv athX g_txpow_ctl frame_subtype` command to retrieve the configured Tx power for control frames.

This capability does not include hardware-generated frames such as CTS, RTS, Ack, Block Ack, and CBF frames. Tx power cannot be controlled in such cases. Frames are eventually transmitted at the least value out of the following values—the user-configured value, hardware capability, conformance test limits (CTL) and regulatory power limitations. The power provided by the user will be in dBm. Some Ctrl frames subtypes can be both FW-generated or HW-generated. This feature does not support such frames (for example, RTS and CTS). The power provided by the user is in dBm and used as a per-chain power similar to the target-power/CTL.

**Table 8-5 Physical layer parameters**

|                                |  |   |   |   |
|--------------------------------|--|---|---|---|
| set_txpow_ctl<br>get_txpow_ctl | iwpriv wifiN set_txpow_ctl <i>frame_subtype</i><br>transmit_power<br><br>iwpriv wifiN get_txpow_ctl<br>frame_subtype | Y | Y | <p>Configure transmit power for control frames for each SSID. The power value configured for a frame can be altered dynamically without any need for restart. The frame subtype is set according to the standard IEEE conventions. For example, to set the transmit power of beacon, enter the following:</p> <pre>iwpriv wifi0 s_txpow_ctl 0x80 8</pre> <p>For example, to obtain the Tx power set for beacon, enter the following command:</p> <pre>iwpriv wifi0 g_txpow_ctl 0x80 wifi0<br/>get_txpow_ctl:8</pre> <p>The transmit power set for a particular frame type can be undone by setting the power to 255. The transmit power value is an 8-bit integer for both direct attach and offload radios. To undo the effects of tx power set for a particular control frame type, use the following command:</p> <pre>iwpriv athN set_txpow_ctl &lt;frame-subtype&gt; 255</pre> |
|--------------------------------|--|---|---|---|

QUALCOMM  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

# 9 Regulatory Compliance of WLAN APs

---

This chapter describes the compliance of WLAN APs with the global regulatory agencies, such as European Telecommunications Standards Institute (ETSI), Federal Communications Commission (FCC), Japan Ministry of Internal Affairs and Communications (MIC) regulations, and also individual countries at all times.

## 9.1 Regulatory Compliance

The WLAN driver code for clients and APs must ensure that transmit powers and operating frequencies meet the requirements of global regulatory agencies (FCC, ETSI, MIC) and individual countries at all times. Similar countries are grouped into regulatory domains, but ultimately the architecture allows any single regulatory rule to differ for any particular country as evolving regulatory law dictates.

A regulatory domain, or regdomain, is the basic definition of a country's rules. Typically, each country has a pair of regdomains, one for 2.4 GHz and one for 5 GHz. Many countries allow channels 1-11 in 2.4 GHz so they might use "FCCA" if the FCC power levels are compliant. The 2.4 GHz regdomain can then be paired with any 5 GHz regdomain to complete the definition.

### 9.1.1 Operation

The channel list, tx power levels, and various flags controlling regulatory operation are contained in the main regulatory data structures. The main structures are as follows:

```
ahCmnRegDomainPairs[]  
    HAL_REG_DOMAIN regDmnEnum; /* 16 bit reg domain pair */  
    HAL_REG_DOMAIN regDmn5GHz; /* 5GHz reg domain */  
    HAL_REG_DOMAIN regDmn2GHz; /* 2GHz reg domain */  
    u_int32_t flags5GHz; /* Requirements flags (AdHoc  
                           disallow, noise floor cal needed, etc) */  
    u_int32_t flags2GHz; /* Requirements flags (AdHoc  
                           disallow, noise floor cal needed, etc) */  
    u_int64_t pscanMask; /* Passive Scan flags which  
                           can override unitary domain  
                           passive scan flags. This  
                           value is used as a mask on  
                           the unitary flags*/  
    u_int16_t singleCC; /* Country code of single country if  
                           a one-on-one mapping exists */
```

This array takes an enumerated regdomain pair, such as FCC3\_FCCA, and points to the single regdomain values in the obvious manner; FCC3 is the 5 GHz domain and FCCA is the 2 GHz

domain. The singleCC entry is a good way to specify a default country when the domain only includes one country that should be set by default.

```
ahCmnAllCountries[]
    HAL_CTRY_CODEcountryCode; /* An internal enumerated country value */
    HAL_REG_DOMAINregDmnEnum; /* A regdomain pair */
    const char*      isoName;      /* A two-letter country code */
    const char*      name;         /* The full country name */
    u_int8_t
        allow11g : 1,
        allow11aTurbo: 1,
        allow11gTurbo: 1,
        allow11ng20: 1, /* HT-20 allowed in 2GHz? */
        allow11ng40: 1, /* HT-40 allowed in 2GHz? */
        allow11na20: 1, /* HT-20 allowed in 5GHz? */
        allow11na40: 1; /* HT-40 allowed in 5GHz? */

    u_int16_t outdoorChanStart; /* The frequency at which channels are to be
considered outdoor */
```

The following is the main country look-up array. The main purpose is the regdomain pair; other values are self explanatory.

```
ahCmnRegDomains[]
    u_int16_t regDmnEnum; /* value from EnumRd table - the regdomain name */
    u_int8_t conformance_test_limit; /* The CTL group of this regdomain */
    u_int64_t dfsMask; /* DFS bitmask for 5Ghz tables */
    u_int64_t pscan; /* Bitmask for passive scan */
    u_int32_t flags; /* Requirement flags (AdHoc disallow, noise
                      floor cal needed, etc) */
    u_int64_t chan11a[BMLEN]; /* 128 bit bitmask for channel/band selection */
    u_int64_t chan11a_turbo[BMLEN]; /* 128 bit bitmask for channel/band selection */
    u_int64_t chan11a_dyn_turbo[BMLEN]; /* 128 bit bitmask for channel/band
selection */
    u_int64_t chan11b[BMLEN]; /* 128 bit bitmask for channel/band selection */
    u_int64_t chan11g[BMLEN]; /* 128 bit bitmask for channel/band selection */
    u_int64_t chan11g_turbo[BMLEN]; /* 128 bit bitmask for channel/band selection
*/
*/
```

The following is the regdomain definition. Transmit power values are written to the EEPROM in three groups—usually FCC, ETSI, and Japan—so the CTL group determines which set of EEPROM values are used. The “bitmask” arrays at the end are “frequency bands”, which are groups of channels that are usually common building blocks that make up the full band channels.

```
regDmn5GhzFreq[]
regDmn2GhzFreq[]
and
regDmn2Ghz11gFreq[]
    u_int16_t    lowChannel; /* Low channel center in MHz */
    u_int16_t    highChannel; /* High Channel center in MHz */
    u_int8_t     powerDfs; /* Max power (dBm) for channel range when using DFS */
    u_int8_t     antennaMax; /* Max allowed antenna gain */
    u_int8_t     channelBW; /* Bandwidth of the channel */
    u_int8_t     channelSep; /* Channel separation within the band */
    u_int64_t    useDfs; /* Use DFS in the RegDomain if corresponding bit is set */
    u_int64_t    usePassScan; /* Use Passive Scan in the RegDomain
                                if corresponding bit is set */
    u_int8_t     regClassId; /* Regulatory class id */
```

The lowChannel and highChannel define contiguous channels to add to the regdomain. The enumerated band names are indicated by comments next to the frequency band definition; for example, /\* F1\_5180\_5240 \*/ is one of the bands that define the UNII-1 frequencies (channels 36-48).

A country regdomain look-up example would be instructive at this point. The United States has been defined in ahCmnAllCountries[] as:

```
{CTRY_UNITED_STATES, FCC3_FCCA, "US", "UNITED STATES", YES, YES, YES, YES, YES,
YES, YES, 5825 }
```

In ahCmnRegDomainPairs[], FCC3\_FCCA uses FCCA for 2.4 GHz as expected:

```
{FCC3_FCCA, FCC3, FCCA, NO_REQ, NO_REQ, PSCAN_DEFER, 0 }
```

In ahCmnRegDomains[], the FCCA definition is seen:

```
{ FCCA, FCC, NO_DFS, NO_PSCAN, NO_REQ,
CHAN_11A_BMZERO
CHAN_11A_BMZERO
CHAN_11A_BMZERO
BM(F1_2412_2462,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1),
BM(G1_2412_2462,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1),
CHAN_TURBO_G_BM(T2_2437_2437,-1, -1, -1, -1, -1, -1, -1, -1, -1, -1)
}
```

The FCC transmit power CTLs are used. The 11g channels use a single frequency band, G1\_2412\_2462.

In regDmn2Ghz11gFreq[], the frequency band is shown to be

```
{ 2412, 2462, 27, 6, 20, 5, NO_DFS, NO_PSCAN, 12 }
```

which is from channel 1-11. That completes the look-up. There are many flags defined along the way.

## 9.1.2 Direct Attach Implementation

### 9.1.2.1 Files

The regulatory module that generates the channel for LMAC and UMAC is located in the HAL module for the direct-attach implementation. This module in HAL generates entries on the LMAC channel list on a per-channel-number and per-channel-mode bases. The channel mode corresponds to operation mode. This channel list is passed and translated to the UMAC channel list for the UMAC to query the available channel for the specified country.

- ah\_regdomain.c

Contains regulatory functions and access functions to the regulatory definitions.

- ah\_regdomain.h

Contains defines for regulatory structures.

### 9.1.2.2 ah\_regdomain\_common.h

Contains defines for regdomains and countries.

### 9.1.2.3 ah\_regdomain\_priv.h (optional)

Contains custom defines for regdomains and countries. May be split to more than one file.

## 9.1.3 Software APIs

### 9.1.3.1 File: ah\_regdomain.c

- Function: ath\_hal\_getwirelessmodes  
Return the mask of available modes based on the hardware capabilities and the specified country code.
- Function: ath\_hal\_ispublicsafetysku  
Return true if the programmed EEPROM regdomain is for the public safety 4.9 GHz band.
- Function: findCountryCode  
Return the country based on the two-letter ISO 3166 country code.
- Function: findCTLByCountryCode  
Return the CTL group for the specified country and band.
- Function: ath\_hal\_getDomain  
Convert Country/RegionDomain to RegionDomain (RD could be region or country code).
- Function: ath\_hal\_init\_channels  
Set up the channel list based on the information in the EEPROM and any supplied country code. EEPROM verification is performed here, and certain regulatory-related access control data used later is also set up.
- Function: ath\_hal\_checkchannel  
Return whether or not the specified channel is allowed based on the current regulatory domain constraints and DFS interference.
- Function: ath\_hal\_getantennareduction  
Return the maximum allowed antenna gain and apply any regulatory domain specific changes.
- Function: ath\_hal\_getantennaallowed  
Return the maximum antenna gain allowed.
- Function: ath\_hal\_getctl  
Return the CTL group from the specified channel from the regulatory table.
- Function: ath\_hal\_getnfcheckrequired

Return whether or not a noise floor check is required in the current regulatory domain for the specified channel.

- Function: `ath_hal_getCurrentCountry`

Return the HAL\_COUNTRY\_ENTRY structure for the current country.

- Function: `ath_hal_set_regdmn_country`

Manually set regdomain, country code, and DFS domain.

- Function: `ath_hal_set_channels`

Copy a list of channels to the HAL channel table.

- Function: `x18_populate_regdomain_tables`

An example of a function used to replace the common regdomain definitions for vendor specific data. This is called at attach time.

## 9.1.4 Offload Regulatory Compliance

The VHT20, VTH40 and VHT80 channel is supported for the direct attach code base. The regulatory module is moved from `ah_regdomain.c` under HAL module to `ol_regdomain.c` under regulatory domain module of host driver. The regulatory module generates the UMAC channel list. The UMAC channel entry includes the maximum transmit power, maximum allowed antenna gain, and CTL group. Those values are passed to the firmware WHAL to limit the transmit power for a specific country through the WMI set channel command.

### 9.1.4.1 Files

- `ol_regdomain.c`

Contains regulatory functions and access functions to the regulatory definitions.

- `ol_regdomain.h`

Contains defines for regulatory structures.

- `ol_regdomain_common.h`

Contains defines for regdomains and countries.

### 9.1.4.2 Software APIs

#### File: `ol_regdomain.c`

- Function: `ol_regdmn_attach`

Regdomain initialization function.

- Function: `ol_regdmn_start`

Regdomain to start to generate a channel list after the firmware ready indication.

- Function `ol_ath_pdev_set_regdomain`

Pass the current regdomain and CTL code to firmware.

- Function: ol\_regdmn\_getWirelessModes  
Return the mask of available modes based on the hardware capabilities and the specified country code.
- Function: ol\_regdmn\_ispublicsafetysku  
Return true if the programmed EEPROM regdomain is for the public safety 4.9 GHz band.
- Function: ol\_regdmn\_findCountryCode  
Return the country based on the two-letter ISO 3166 country code.
- Function: ol\_regdmn\_findCTLByCountryCode  
Return the CTL group for the specified country and band.
- Function: ol\_regdmn\_getDomain  
Convert Country/RegionDomain to RegionDomain (RD could be region or country code).
- Function: ol\_regdmn\_init\_channels  
Set up the channel list based on the information in the EEPROM and any supplied country code. EEPROM verification is performed here, and certain regulatory-related access control data used later is also set up.
- Function: ol\_ath\_get\_CurrentCountry  
Return the country based on the two-letter ISO 3166 country code.
- Function: ol\_ath\_set\_country  
Manually set country code.
- Function: ol\_ath\_set\_regdomain  
Manually set regdomain.
- Function: ol\_ath\_get\_dfsdomain  
Return DFS domain code.

## 9.2 Offload FCC regulatory rules to firmware

This section describes the functionality for offloading FCC/US regulatory rules to firmware. Regulatory offload to firmware is to secure the of FCC/US rules so the device operates with them. This enables OEM's to allow HLOS replacement to support after-market usage and upgrading/replacement of the router software. The following are the basic requirements:

- Prevent operation on channels 12-14 in 2.4 GHz and any invalid channels on 5 GHz.
- Prevent power levels greater than maximum allowed by FCC.
- Allow replacement of the HLOS, but keep the regulatory parameters.
- Check flags like DFS, Quarter rate/half rate etc for proper usage.

## Design changes

- The Country Code/Regulatory Domain BITS to be stored in the OTP, only accessible by Firmware.
- For more security, the FCC/US specific Regulatory Rules are stored in Firmware.
- HOST would still send initial Regulatory Rules to FW using command WMI\_SCAN\_CHAN\_LIST\_CMDID.
- Firmware will check the validity of Regulatory Rules sent by HOST in case the Current Country is US/FCC.
- When a host sets a certain channel in case of FCC/US, firmware uses these tables to determine two aspects.
  - a. Target power as per the regulatory rules. (This regulatory target power is one of the max limits of target power as there are boards level constraints which the halphy code currently maintains)
  - b. Allowing to operate on a certain channel.
- In case the Current Country is US/FCC, Firmware will check the validity of Channels in all host originating commands like WMI\_START\_SCAN, WMI\_VDEV\_START, etc.
- If a violation occurs during the firmware check, the firmware will report error using WMI\_REG\_ERROR\_EVENTID.

The current design will support only FCC/US country code rules to be stored by Firmware due to memory limitations. This can be extended in future to support other important regulatory domain databases. The firmware extracts the country code/reg domain code either from OTP or calibration data.

The format for OTP Regulatory Code is as follows:

- Bit-15 indicates whether its Country code or Domain code.
- Lower 15-bits indicate info - for Country code or Domain code.

| * Bit-15 | Bit-14...Bit0 |  |
|----------|---------------|--|
| 0        | xxxxxx        | -> Bit14-0 is a Regulatory domain pair |
| 1        | xxxxxx        | -> Bit14-0 is a Country code.          |

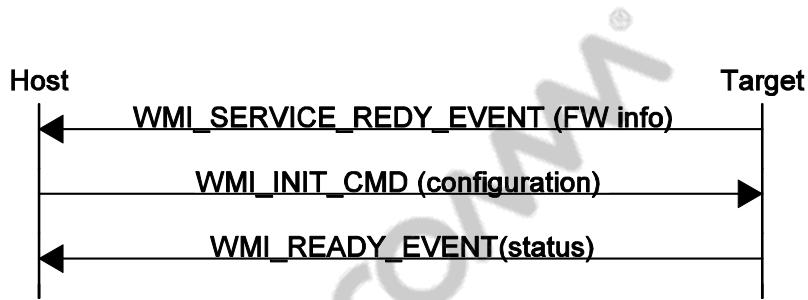
Once country code is extracted, it is converted into reg domain pairing using country\_code\_to\_reg\_dmn mapping, using reg\_dmn\_pair value, individual 5g and 2g reg domain ids can be obtained. Depending on these reg-domain ids, regulatory rules are populated from regdomain structure to global regulatory structure. This can be used for any subsequent checks.

## Host/Target Interaction

This subsection describes the various interaction between the host and target for this feature. FW extracts regulatory ID/Country code from OTP/Cal data and passes it up to the host via WMI\_SERVICE\_READY event. A regulatory Domain module on Host, then uses the information to construct an initial channel list. The list includes a list of channels along with the necessary regulatory information for each channel.

The regulatory information about each channel includes frequency, allowed modes (HT20/40/80, VHT20/40/80), max transmit power, passive/dfs, etc. The list is constructed based on regulatory domain/ country-code values specified in the OTP/Cal data.

- Once the country code is read by firmware from OTP, it is sent to host using HAL\_REG\_CAPABILITIES->EEPROM\_rd. during the WMI\_SERVICE\_READY command. This value is same as programmed in OTP.
- Host will check the WMI\_SERVICE\_READY and uses the country code. Once valid value is set from firmware, host will not accept any country code value from UCI/iwpriv.



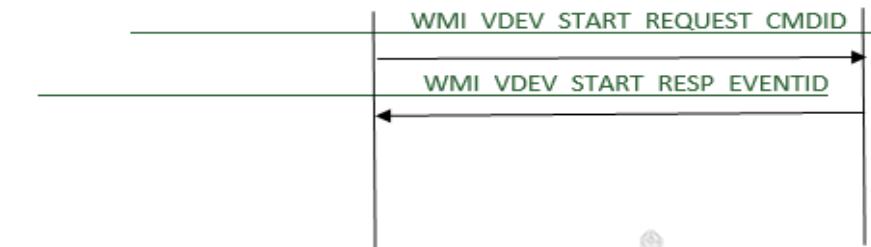
## WMI commands

There are many WMI commands currently making use of channel and power information from host. Checks are added for the validity and error response events are sent to host for any violation. This is independent of AP mode. The following are WMI commands:

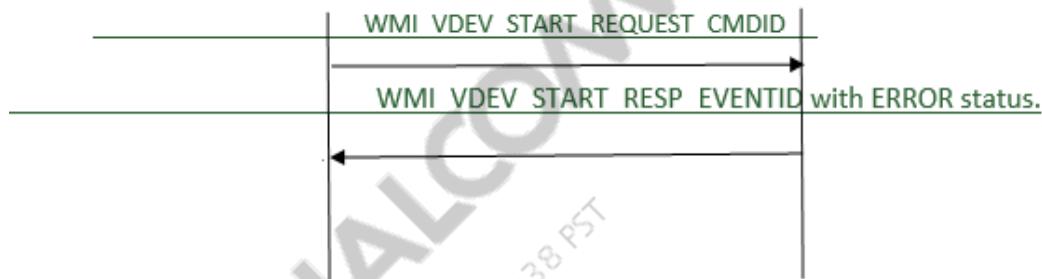
| WMI Command                      | Response from host          |
|----------------------------------|-----------------------------|
| WMI_START_SCAN_CMDID             | No                          |
| WMI_SCAN_CHAN_LIST_CMDID         | No                          |
| WMI_PDEV_SET_CHANNEL_CMDID       | No                          |
| WMI_VDEV_START_REQUEST_CMDID     | WMI_VDEV_START_RESP_EVENTID |
| WMI_CSA_OFFLOAD_CHANSWITCH_CMDID | No                          |
| WMI_SET_CHANNEL_MAX_POWER        | No                          |
| WMI_PDEV_PARAM_TXPOWER_LIMIT2G   | No                          |
| WMI_PDEV_PARAM_TXPOWER_LIMIT5G   | No                          |
| WMI_PDEV_SET_REGDOMAIN_CMDID     | No                          |

In case of any violation for FCC/US country code on comparison with regulatory values maintained in local regulatory data base, Firmware uses local data base values for max-power and channel information and also sends any error response in response event.

## Normal scenario



## Error scenario



The firmware sends VDEV\_START\_RESP or VDEV\_RESTART\_RESP event with status as WMI\_VDEV\_START\_CHAN\_INVALID.

```

typedef enum {
    /* Event response of START CMD */
    WMI_VDEV_START_RESP_EVENT = 0,
    /* Event response of RESTART CMD */
    WMI_VDEV_RESTART_RESP_EVENT,
} WMI_START_EVENT_PARAM;

typedef enum {
    WMI_VDEV_START_OK = 0,
    WMI_VDEV_START_CHAN_INVALID,
} WMI_VDEV_START_STATUS;

typedef struct {
    /** unique id identifying the VDEV, generated by the caller */
    A_UINT32 vdev_id;
    /** requestor id that requested the VDEV start request */
    A_UINT32 requestor_id;
    /* Response of Event type START/RESTART */
    WMI_START_EVENT_PARAM resp_type;
    /** status of the response */
    WMI_VDEV_START_STATUS status;
} wmi_vdev_start_response_event;
  
```

## Host Response to WMI\_VDEV\_START\_REQUEST\_CMDID with error status.

Host does not call ieee80211\_vap\_join() to move VAP state machine to join and keep the VAP in the IEEE80211\_S\_INIT state in ol\_vdev\_wmi\_event\_handler(ol\_scn\_t scn, u\_int8\_t \*data, u\_int16\_t datalen, void \*context) handler when the invalid channel response message is received.

The following are the data structures:

```

struct regulatory_rule {
    u16 start_freq;
    u16 end_freq;
    u8 max_bw;
    u8 max_power;
    u8 channelSep;
    u32 flags;
};

struct regdomain {
    u8 regdomain_id;
    u8 ctl;
    u32 rules_bitmap;
};

struct regulatory_rule FCC_RULE1 = {
    .start_freq = 5170,
    .end_freq = 5250,
    .max_bw = 80,
    .NOPSCAN | NO_DFS,
    .max_power = 17
}

struct regulatory_rule FCC_RULE2 = {
    .start_freq = 5170,
    .end_freq = 5250,
    .max_bw = 80,
    .NOPSCAN | NO_DFS,
    .max_power = 23
}
.
.
.

struct regdomain FCC1 = {
    .regdomain_id = FCC1;
    .num_reg_rules = 3;
    .rule_ptr = {
        FCC_RULE1,
        FCC_RULE5,
        FCC_RULE10
    };
}

struct country_code_to_reg_dmn {
    uint16_t country_code;
    uint16_t reg_dmn_pair;
    const char *alpha2;
};

```

```

struct reg_dmn_pair {
    uint16_t reg_dmn_pair;
    uint16_t reg_dmn_5ghz;
    uint16_t reg_dmn_2ghz;
};

typedef struct channel {
    /* primary 20 MHz channel frequency in mhz */
    A_UINT16 mhz;
    /* Center frequency 1 in MHz*/
    A_UINT16 band_center_freq1;
    /* Center frequency 2 in MHz - valid only for 11acvht 80plus80 mode*/
    A_UINT16 band_center_freq2;
    /* phy mode */
    WLAN_PHY_MODE phy_mode;
    A_UINT8 max_reg_power;
    A_UINT8 antenna_max;
    A_UINT8 flags;
} channel_t

```

### Error checking on channel

In case of vdev\_start, vdev\_restart, scan Chan list, following must be satisfied.

- Operating frequency (MHz) must be valid channel, it must be in one of the supporting frequency bands. For phymodes of 20, 40, 80,160 MHz .this must be a valid 20 MHz channel.
- Center frequency must be a valid one based on bandwidth used. This can be derived by starting from band start frequency with required offset and incrementing by bandwidth from phymode (for 20, 40, 80 quarter and half channel).check channelsep field to check if quarter and half rates are possible in the band.
- Center freq 2 which is used in case of VHT80\_80 and VHT160, In this case both must be valid center frequencies for 80 mode , difference between center frequencies must be 80 in case of VHT160 and >80 in case of VHT80\_80.

If any of this conditions are not met vdev\_start\_response reports invalid channel, scan discards that particular channel. In addition, power levels are set to the maximum permissible values for that band if a value higher than the valid maximum value is entered.

## 9.3 Preserve regulatory settings after HLOS replacement

The functionality to support regulatory offload to firmware enables securing the FCC/US rules for the device operates with these rules. This enables OEMs to allow HLOS replacement to support after-market usage and upgrade or replacement of the router software. The following are the salient benefits of this feature:

- Prevent operation on channels 12-14 in 2.4GHz and any invalid channels on 5GHz.
- Prevent power levels greater than maximum allowed by FCC.
- Allow replacement of the HLOS with preservation of the regulatory parameters.
- Check flags such as DFS, quarter rate, and half rate for proper usage.

This feature helps in maintaining regulatory information for US country and relevant FCC domains. With the regulatory data stored in Firmware, this functionality prevents misuse of power, prevents antenna gain or invalid operation on DFS channels, and disallows unsupported frequencies. Support is provided only for US country code rules to be stored by Firmware due to memory limitations. When any problem is observed with the feature, enter the following command to check the current regulatory that is used:

```
wifitool athX beeliner_fw_test 132 1
```

Consider a scenario in which power level is hard-coded to more than 35 in vdev\_start for US reg-domains, and check if power value is accepted. Power value is given as 35 in vdev\_start for channel 36. The maximum reg power is changed to 30 after regulatory check; this regulatory power is the maximum value in that band for US (843).

For US reg-domain, if an invalid central frequency is hardcoded for a HT80 mode, an error response is sent because center frequency is not valid for this mode. If the EEPROM reg-domain value is hard-coded to US and is different from the value in PDEV\_REGDOMAIN command, the AP continues to operate in US domain. For non-US pairs, the firmware code is not used. Quarter rate is not applied to 5 GHz channels and it is applied for public safety (PS) reg-domains only.

## 9.4 Wi-Fi LTE-U operation

This section describes the Wi-Fi support for LTE-U operation on unlicensed bands. The requirements on the Wi-Fi Host and firmware are illustrated with flow diagrams. This section also describes the AP link library to Wi-Fi host interface specifications and Wi-Fi host to firmware interface specifications.

LTE-Unlicensed (henceforth referred to as LTE-U) is a new radio access technology that has been proposed for providing carrier-grade wireless service in the 5 GHz unlicensed band in certain countries such as U.S. and Korea (for example, UNII radio bands in the US covering 5.15 GHz – 5.825 GHz regulated by the FCC). Currently, Wi-Fi (WLAN that uses the IEEE 802.11 standard) has been the most popular choice for radio access in the unlicensed space. However, recent studies have revealed that LTE technology, originally envisioned for cellular operation in the licensed band, has significant performance gain over Wi-Fi when its operation is extended to the unlicensed band. The main advantages for LTE-U over Wi-Fi as an access technology stems from better link performance, medium access control, mobility management and excellent coverage. These benefits combined with the vast amount of available spectrum (> 400MHz) in the 5GHz band make LTE-U a promising radio access technology in the unlicensed arena.

Because Wi-Fi devices are already widespread in the 5GHz unlicensed band, it is necessary for newly deployed LTE-U Small Cell (SC) to protect and coexist seamlessly with the Wi-Fi ecosystem. Also, different LTE-U operators may occupy the same spectrum in the unlicensed band to provide data services to their users. Such an unplanned and unmanaged deployment of LTE-U SCs (femtocells, picocells) may result in excessive RF interference to the existing co-channel Wi-Fi and other operator LTE-U nodes. It is therefore critical for LTE-U SCs to choose the best operating channel while minimizing the interference caused to nearby Wi-Fi and LTE-U networks. QCA9880 chipset is used for AP scan or medium utilization (MU) scan purpose for this feature. QCA9880 is operating in VHT80, 3x3 mode.

## 9.4.1 LTE-U requirements

This subsection describes the WLAN host and firmware requirements for Wi-Fi LTE-U support:

### 9.4.1.1 AP scan

On the AP side, measurements are done on the beacon, probe response frames from other APs. Upon power up, initialization and configuration, the SC LTE-U CrM module requests the co-located Wi-Fi AP to scan all candidate channels. The AP scans each channel sequentially for beacons (when LTE-U nodes from the same operator are guaranteed to be OFF) and sends the measurements to the SC LTE-U CrM module to choose the cleanest operating channel. Once LTE-U is ON, the SC LTE-U CrM periodically requests the AP to perform channel scanning on the operating and non-operating channels every few tens of seconds. The Wi-Fi module responds with the scan report comprising of all BSSIDs found in the requested channel list.

### 9.4.1.2 Medium utilization

The SC LTE-U cognitive radio module (CrM) requests the co-located Wi-Fi AP to monitor a particular channel for all decodable Wi-Fi packets and report the medium utilization. The request is sent to the Wi-Fi module along with time duration for which channel has to be monitored and other algorithm related inputs required for the MU computation. The Wi-Fi module responds with the MU report depending on the chosen MU computation algorithm.

### 9.4.1.3 Host software guidelines

- For AP scan, the application uses ioctl IEEE80211\_IOCTL\_SETPARAM to set scan parameters such as probe delay, and idle time that cannot be set using the scan request itself.
- The application then issues SIOCSIWSCAN ioctl to initiate the scan. This has the channels to be scanned and scan parameters such as dwell time and scan type (active/passive).
- If any errors occur while initiating the AP scan in the driver, certain IOCTL error values are returned. The failure reason and corresponding IOCTL error code values returned are as listed as follows:
  - IOCTL error code returned if netdevice is not up and running: -EINVAL
  - IOCTL error code returned if previous scan is in progress: -EBUSY
  - IOCTL error code returned if failure occurs in copying scan parameters from user space:-EFAULT

- IOCTL error code returned if failure occurs in allocation of scan parameters in driver: - ENOMEM
- If no errors were encountered in the initiation of AP scan, the host driver sends a WMI command to the firmware passing the channel list and the scan parameters.
- The host driver flushes the old scan table and makes a new one from the beacons and probe responses received.
- The host driver receives the scan completion WMI event from the firmware and indicates the same to the application using SIOCGIWSCAN wireless event. This event indicates whether the scan was successful or not.
- The application can then query the scan results using the SIOCGIWSCAN ioctl.
- For MU measurement, the application issues ioctl IEEE80211\_IOCTL\_DBGREQ with type set to IEEE80211\_DBGREQ\_MU\_SCAN. This has the channel on which MU has to be computed and MU params such as the duration and the algorithm to use.
- The host driver sends a WMI command to the firmware to switch to the requested channel, if the requested channel is not same as the current channel.
- The host driver then sends another WMI command to start MU computation for the requested duration.
- In case of any issues encountered in the course of initiating an MU scan by the host driver, following are the probable reasons for error and the corresponding IOCTL error code returned if that issue is encountered:
  - IOCTL error code returned if MU channel is invalid or the ioctl is used when radio is not in LTEu mode: -EINVAL
  - IOCTL error code returned if a previous MU scan is in progress: -EBUSY
  - IOCTL error code returned if WMI command send failed because some resource couldn't be allocated: -ENOMEM
  - IOCTL error code returned if algo type is zero or exceeds maximum value of 15 and if duration of scan is 0: -EINVAL.
  - In case of any other error in sending WMI command: -1
- If MU scan was successfully initiated, the host driver received the MU report WMI event from the firmware which has the computed MU value. The same is notified to the application using IWEVGENIE wireless event with type set to IEEE80211\_EV\_MU\_RPT.

#### 9.4.1.4 Target firmware guidelines

- Wi-Fi FW receives WMI command from Wi-Fi host to start scan with channel no. and scan parameters passed from the application.
- At this point, the following two implementations. Either of them is selected, depending on an input config passed from host.
  - Wi-Fi FW starts scan immediately upon receiving the start scan command. The scan continues for the duration mentioned in scan parameters.

- Wi-Fi FW stays idle after setting the scan parameters. The actual scan starts in interrupt handler context after the GPIO pin is toggled. Scanning continues till the scan duration mentioned in input parameters is over.
- Wi-Fi FW switches to channel in the channel list and dwells on the channel for listening to the beacons and/or sending probe request depending on active/passive scan configuration and forwards to Wi-Fi host.
- At the end of the scan, Wi-Fi FW sends a scan event indicating successful scan completion.
- Wi-Fi FW receives WMI command for set channel. Wi-Fi FW switches to this particular channel.
- Wi-Fi FW receives WMI command to start monitoring the channel with MU duration, and other inputs needed for MU computation.
- At this point, the following two implementations can occur. Either of them is selected, depending on an input flag passed from host.
  - Wi-Fi FW starts MU measurement immediately upon receiving the start MU command. The MU measurement continues for the duration mentioned in MU start command.
  - Wi-Fi FW stays idle after setting the MU parameters. Actual MU measurement starts in interrupt handler context once the GPIO pin is toggled. MU computation continues until the MU duration mentioned in input params is over.
- FW decodes the preamble and MAC header if possible and get the RSSI, frame duration, BSSID, data/ack info from the sniffed packets and compute the MU.
- After the monitoring duration is over, Wi-Fi FW sends a MU success event together with the MU report.

## 9.4.2 WLAN driver design

This subsection describes the WLAN driver design enhancements for LTE-U support:

### 9.4.2.1 WLAN driver mode of operation

- Wi-Fi firmware is configured in the AP mode itself for LTEU support with few modifications. No new mode is being defined for this feature explicitly. To enable this mode, user to add ‘lteu\_support=1’ for wifi0 interface and ‘lteu\_support=2’ for wifi1 interface in load params during loading the driver. The lteu\_support load param is only applicable for QCA9880 11ac offload wifi radio.
- During load time, host informs firmware that firmware must enable LTEU support. On the host, a new parameter is added in wlan\_target\_resource\_config structure for this purpose and is set when required.

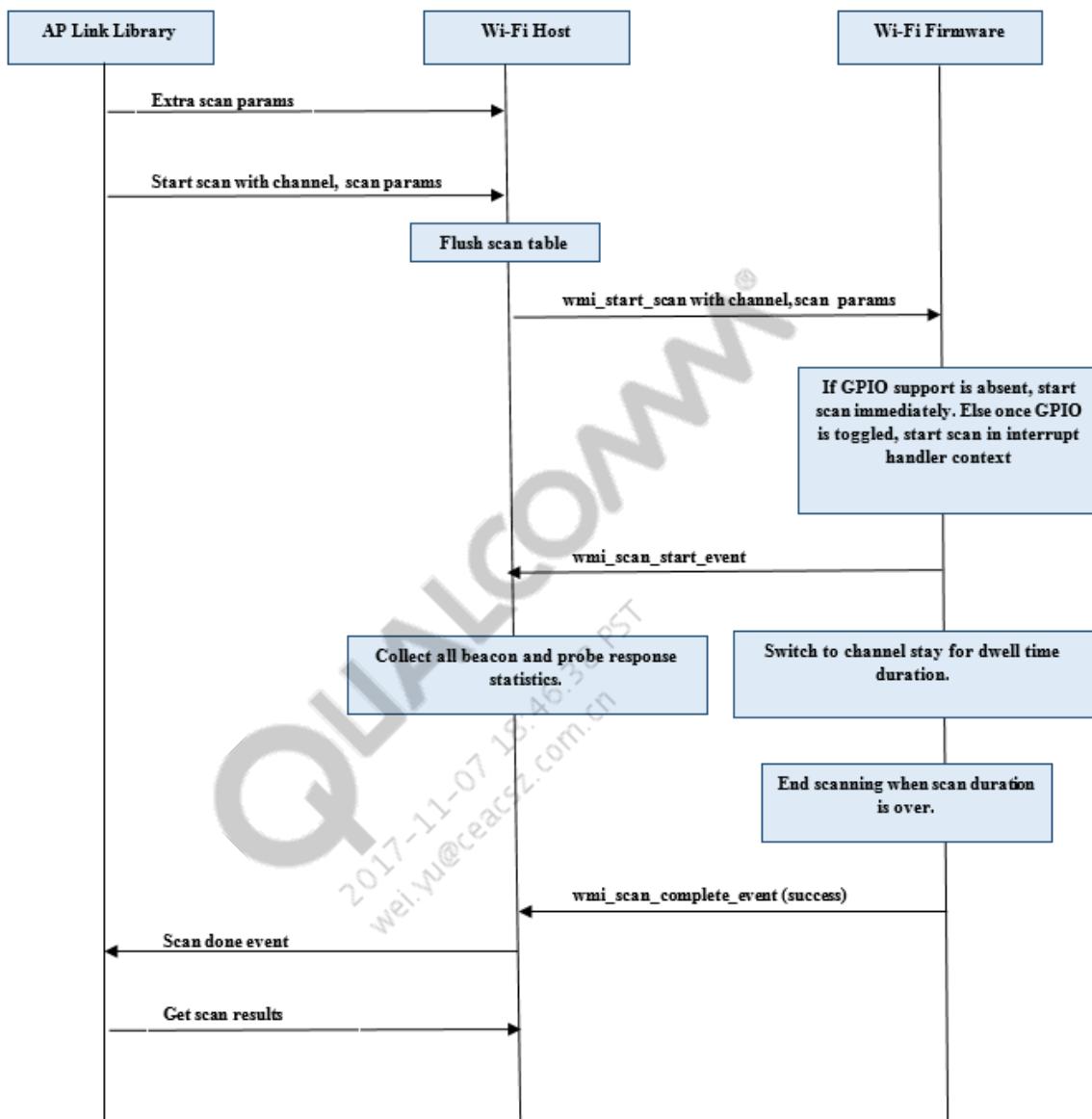
```
typedef struct {
    ...
    ...
    fw_feature_bitmap;           // FW_FEATURE_LTEU_SUPPORT
    0x0001
} wmi_ext_resource_config;
```

The LTE-U support mode is added in firmware in wlan\_target\_resource\_config structure.

```
typedef struct {  
    ...  
    ...  
    lteu_support_config;  
} wlan_target_resource_config;
```

- While loading the LTEU support mode, host implicitly reduces the maximum number of peers supported to free up memory on the firmware.
- In LTEU support mode, for MU measurements, FW sets the promiscuous bit in the RX-filter to enable sniffing of all packets.
- For MU scan, QCA9880 will operate in VHT80 mode by default. In case, the operating channel is set to such that VHT80 band is not possible (such as CH132/136), QCA9880 attempts to lower the bandwidth to HT40. If HT40 operation is also not possible (such as CH165), QCA9880 lowers the bandwidth further and operate on HT20.

#### 9.4.2.2 AP scan



### Interaction between AP link library and host software interface

The existing scan framework is used for AP scan with few modifications:

- **Scan start :** The application calls SIOCSIWSCAN icctl. The following parameters are passed in struct iw\_scan\_req:
  - scan\_type specifies whether it is requesting active scan or passive time
  - num\_channels is set to number of channels to be scanned
  - flags parameter has IW\_SCAN\_THIS\_FREQ set
  - min\_channel\_time and max\_channel\_time specify the scan duration
  - channel\_list[] contains the channels to be scanned

- **Scan event** : On the completion of the scan the driver sends SIOCGIWSCAN event to the application. This is just an indication that the scan is complete, it doesn't contain any scan results. The flag indicates whether the scan completed successfully or not - IEEE80211\_REASON\_COMPLETED indicates success and IEEE80211\_REASON\_RUN\_FAILED indicates error.
- **Scan results** : The application uses SIOCGIWSCAN ioctl to get the scan results. The scan results are returned in the buffer allocated by the application. The results are delivered in the same format as they are currently done in the driver. Additionally, fields are included to report timestamp and sequence of beacons or probe response frames. It is reported as a wireless event(iw\_event) of type IWEVCUSTOM. The additional field that is reported per scan table entry is of format: timestamp = %llu and sequence = %u.
- **Scan params** : Additional scan params such as probe delay, rest time, idle time and repeat probe time can be configured individually using the IEEE80211\_IOCTL\_SETPARAM ioctl.

### Interaction between host software and target firmware interface

The existing scan framework is used for AP scan with few modifications.

- **Scan start** : To be done using existing WMI\_START\_SCAN\_CMDID with appropriate scan params from host to firmware. Existing scan ID is used to map the scan start and responses between host and firmware. For multiple probe requests generation, the repeat\_probe\_time and probe\_spacing\_time parameters can be used.

```

typedef struct {
    A_UINT32 scan_id;
    A_UINT32 scan_req_id;
    A_UINT32 vdev_id;
    A_UINT32 scan_priority;
    A_UINT32 notify_scan_events;
    A_UINT32 dwell_time_active;
    A_UINT32 dwell_time_passive;
    A_UINT32 min_rest_time;
    A_UINT32 max_rest_time;
    A_UINT32 repeat_probe_time;
    A_UINT32 probe_spacing_time;
    A_UINT32 idle_time;
    A_UINT32 max_scan_time;
    A_UINT32 probe_delay;
    A_UINT32 scan_ctrl_flags;
    /**
     * TLV (tag length value ) paramerters follow the scan_cmd
     * structure.
     *
     * TLV can contain channel list, bssid list, ssid list and
     * ie. the TLV tags are defined above;
     */
}

```

```
    } wmi_start_scan_cmd;

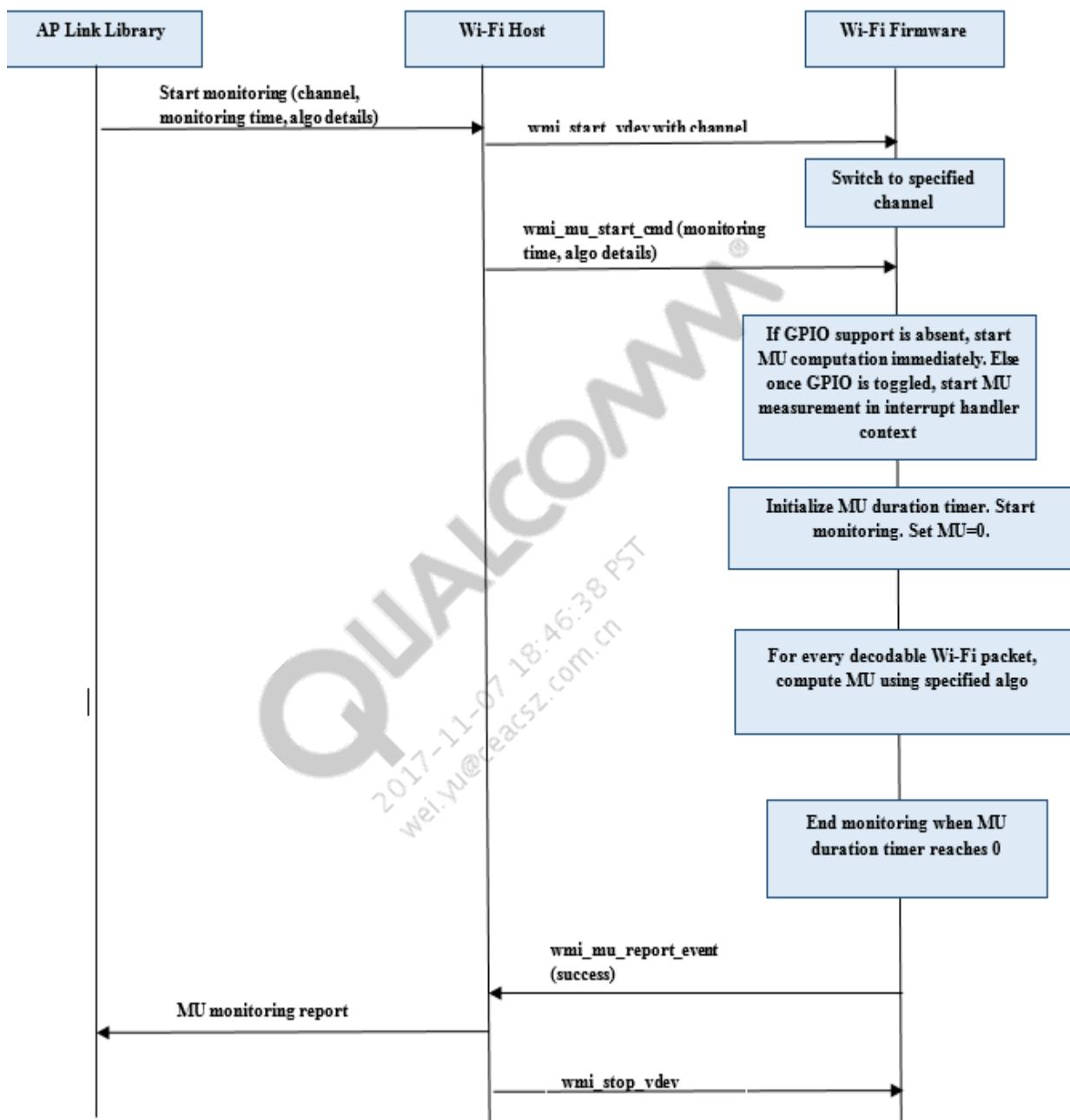
■ Scan event : Use the following existing WMI_SCAN_EVENTID to indicate scan events.

enum wmi_scan_event_type {
    WMI_SCAN_EVENT_STARTED=0x1,
    WMI_SCAN_EVENT_COMPLETED=0x2,
    WMI_SCAN_EVENT_BSS_CHANNEL=0x4,
    WMI_SCAN_EVENT_FOREIGN_CHANNEL = 0x8,
    WMI_SCAN_EVENT_DEQUEUED=0x10,           /* scan request got dequeued */
    WMI_SCAN_EVENT_PREEMPTED=0x20,          /* preempted by other high
priority scan */
    WMI_SCAN_EVENT_START_FAILED=0x40,        /* scan start failed */
    WMI_SCAN_EVENT_GPIO_TIMEOUT=0x80, /* in case gpio timed out/
scan got

interrupted due to some reason */

};
```

#### 9.4.2.3 Medium utilization scan



## Medium utilization algorithm

All RSSI mentioned here are in dBm. In Wi-Fi FW, per-PPDU RSSI is converted from dB to dBm before using the RSSI value for MU computation. The following are the different algorithms for MU computation. One of the algorithms is selected at runtime.

- Basic MU computation:
  - For every decodable Wi-Fi packet, calculate preamble RSSI (whether the energy is above or below CCA-ED) and extract packet duration (in  $\mu$ s).

- Record RSSI of the detected packet. Increment MU counter as follows

If RSSI > RSSI\_Thr1, then MU+=W1\*PKT\_Duration

If RSSI\_Thr2 < RSSI <= RSSI\_Thr1, MU+= W2\* PKT\_Duration.

If RSSI\_Thr3 < RSSI <= RSSI\_Thr2, then MU+= W3\*PKT\_Duration.

Where W1, W2, and W3 take values in [0, 1]

MU = MU/Monitoring time

- Enhanced MU computation
- TA Based MU computation

### Interaction between AP link library and host software interface

- **Start MU:** The application calls the private ioctl IEEE80211\_IOCTL\_DBGREQ. The arguments are passed in struct ieee80211req\_athdbg. A new cmd IEEE80211\_DBGREQ\_MU\_SCAN is added for MU scan.

```
enum {
    IEEE80211_DBGREQ_SENDADDBA          = 0,
    IEEE80211_DBGREQ_SENDDELBA          = 1,
    ...
    IEEE80211_DBGREQ_TR069              = 24, /* to be used for tr069 */
    IEEE80211_DBGREQ_MU_SCAN            = 47, /* do a MU scan */
};
```

A new typedef ieee80211req\_mu\_scan\_t is added to the union data of struct ieee80211req\_athdbg for passing the scan parameters.

```
typedef enum {

    MU_ALGO_1 = 0x1, /* Basic binning */
    MU_ALGO_2 = 0x2, /* Enhanced binning */
    MU_ALGO_3 = 0x4, /* Enchanced binning including accounting for hidden nodes */
    MU_ALGO_4 = 0x8, /* TA based MU calculation */
} mu_algo_t;

typedef struct {

    u_int8_t      mu_req_id;           /* MU request id */
    u_int8_t      mu_channel;          /* IEEE channel number on which to do MU scan */
    mu_algo_t     mu_type;             /* which MU algo to use */
    u_int32_t     mu_duration;         /* duration of the scan in ms */
    u_int32_t     lteu_tx_power;        /* LTEu Tx power */
}
```

```

        u_int32_t    mu_rssi_thr_bssid; /* RSSI threshold to account for
active APs */
        u_int32_t    mu_rssi_thr_sta;      /* RSSI threshold to account for
active STAs */
        u_int32_t    mu_rssi_thr_sc;       /* RSSI threshold for WCUBS
detection */
        u_int32_t    home_plmnid;         /*to be compared against plmn id to
distinguish between
                                         *same and different operator WCUBs*/
                                         /* alpha for num active
bssid calculation */
} ieee80211req_mu_scan_t;

struct ieee80211req_athdbg {
    u_int8_t cmd;
    u_int8_t dstmac[IEEE80211_ADDR_LEN];
    union {
        int param[4];
        ieee80211_rrm_beaconreq_info_t bcnrpt;
        ...
        ieee80211req_tr069_t             tr069_req;
        ieee80211req_mu_scan_t          mu_scan_req;
    } data;
};

```

- **Report MU:** On completion of the scan, the driver sends IWEVGENIE event to the application to report the MU. A new custom event IEEE80211\_EV\_MU\_RPT is added for passing the MU scan results.

```

enum {
    IEEE80211_EV_SCAN_DONE,
    IEEE80211_EV_CHAN_START,
    ...
    IEEE80211_EV_WAPI,
    IEEE80211_EV_MU_RPT,
};

```

A new struct event\_data\_mu\_rpt is defined for passing the results of the MU scan.

```

#define MU_MAX_ALGO          4
#define MU_DATABASE_MAX_LEN   32
#define LTEU_MAX_BINS         10
typedef enum {

```

```

        MU_STATUS_SUCCESS ,
        MU_STATUS_BUSY_PREV_REQ_IN_PROG ,
        MU_STATUS_INVALID_INPUT ,
        MU_STATUS_FAIL_BB_WD_TRIGGER ,
        MU_STATUS_FAIL_DEV_RESET ,
        MU_STATUS_FAIL_GPIO_TIMEOUT ,
    } mu_status_t;
typedef enum {
    DEVICE_TYPE_AP , DEVICE_TYPE_STA ,
    DEVICE_TYPE_SC_SAME_OPERATOR,
    DEVICE_TYPE_SC_DIFF_OPERATOR,
}mu_device_t;
typedef struct{
    /* specifying device type(AP/STA/SameOPClass/DiffOPClass)for
    each entry of the MU database*/
    mu_device_t mu_device_type;
    /* specifying BSSID of each entry */
    u_int8_t      mu_device_bssid[IEEE80211_ADDR_LEN];
    /* Mac address of each entry */
    u_int8_t      mu_device_macaddr[IEEE80211_ADDR_LEN];
    /* average packet duration for each device in micro secs to
    avoid decimals */
    u_int32_t     mu_avg_duration;
    /* average rssi recorded for each device */
    u_int32_t     mu_avg_rssi;
    /* percentage of medium utilized by each device */
    u_int32_t     mu_percentage;

}mu_database;

struct event_data_mu_rpt {
    u_int8_t      mu_req_id;                      /* MU request id,
copied from the request */
    u_int8_t      mu_channel;                     /* IEEE channel number
on which MU was done */
    mu_status_t   mu_status;                      /* whether the MU scan
was successful or not */
    /* The aggregate MU computed by basic binning,enhanced binning and
hidden node algo*/
    u_int32_t     mu_total_val[ MU_MAX_ALGO -1 ];
}

```

```

        u_int32_t          mu_num_bssid;           /* number of active
BSSIDs*/
        u_int32_t          mu_actual_duration; /* time in ms for which the
MU scan was done */
        /* The MU computed by the hidden node algo, reported on a per bin
basis */
        u_int32_t          mu_hidden_node_algo[LTEU_MAX_BINS];
        /* number of active TA entries in the database */
        u_int32_t          mu_num_ta_entries;
        mu_database mu_database_entries[MU_DATABASE_MAX_LEN];

    };
}

```

### Interaction between host software and target firmware interface

- **Set channel for MU:** To be done using existing WMI\_VDEV\_START\_REQUEST\_CMDID. This command takes care of setting the channel center frequency and bandwidth for which MU is to be computed. This command has to be sent from host to firmware everytime a channel change is required. WMI\_VDEV\_STOP\_CMDID is expected before WMI\_VDEV\_START\_REQUEST\_CMDID if the vdev is already operational on some other channel.
- **Set MU params:** A new WMI command and structure are introduced for setting MU monitoring parameters and starting MU measurements:

```

WMI_MU_CAL_START_CMDID
typedef struct {
    A_UINT32 mu_request_id;
    A_UINT32 mu_duration;
    MU_ALGO_TYPE mu_type;
    A_UINT32 rssi_thr_bssid;
    A_UINT32 rssi_thr_sta;
    A_UINT32 rssi_thr_sc;
    A_UINT32 alpha_num_bssid;
    A_UINT32 plmn_id;
} wmi_mu_start_cmd;

typedef enum {
    MU_BASIC_ALGO = 0x1,
    MU_PER_BSSID_ALGO = 0x2 ,
    MU_HIDDEN_NODE _ALGO = 0x4,
    MU_PER_TA_ALGO = 0x8,
} MU_ALGO_TYPE;

```

- **Report MU utilization:** A new WMI event and structure are introduced for MU reporting:

```

-
    WMI_MU_REPORT_EVENTID
-
    #define MAX_ALGO_TYPE 4
-
    #define MAX_MU_DB_ENTRIES 32
-
    #define LTEU_MAX_BINS 10

-
    typedef struct {
        A_UINT32 entry_type;
        wmi_mac_addr bssid_mac_addr;
        wmi_mac_addr ta_mac_addr;
        A_UINT32 avg_duration_us;
        A_UINT32 avg_rssi_dbm;
        A_UINT32 mu_percent;
    } wmi_mu_db_entry;

-
    typedef struct {
        A_UINT32 mu_request_id;
        MU_STATUS_REASON status_reason;
        A_UINT32 total_mu[MAX_ALGO_TYPE-1];
        A_UINT32 num_active_bssid;
        A_UINT32 hidden_node_mu[LTEU_MAX_BINS];
        A_UINT32 num_TA_entries;
        wmi_mu_db_entry mu_entry[MAX_MU_DB_ENTRIES];
    }wmi_mu_report_event;

typedef enum {
    MU_SUCCESS,
    MU_BUSY, /* Not used anymore. Here only for backward compatibility */
    MU_INVALID, /* Not used anymore. Here only for backward compatibility */
    MU_BUSY_PREV_REQ_IN_PROG, /* Previous MU request is in progress */
    MU_INVALID_INPUT,
    MU_FAIL_BB_WD_TRIGGER,
    MU_FAIL_DEV_RESET,
}

```

```

        MU_FAIL_GPIO_TIMEOUT /* gpio timed out /MU got
interrupted due to some reason */
}MU_STATUS_REASON;

```

## 9.4.3 LTE-U configuration

### 9.4.3.1 Interaction between AP link library and host software interface

**Start MU:** The application calls the private ioctl IEEE80211\_IOWR\_DBGREQ. The arguments are passed in struct ieee80211req\_athdbg.

A new cmd IEEE80211\_DBGREQ\_LTEU\_CFG is added.

```

enum {
    IEEE80211_DBGREQ_SENDADDBA = 0,
    IEEE80211_DBGREQ_SENDELBA = 1,
    ...
    IEEE80211_DBGREQ_MU_SCAN = 47, /* do a
MU scan */
    IEEE80211_DBGREQ_LTEU_CFG = 48, /* LTEu
specific configuration */
};

```

A new typedef ieee80211req\_lteu\_cfg\_t is added to the union data of struct ieee80211req\_athdbg for passing the configurable parameters.

```

#define LTEU_MAX_BINS 3

typedef struct {
    u_int8_t lteu_gpio_start; /* start
MU/AP scan after GPIO toggle */
    u_int8_t lteu_num_bins; /* no. of
elements in the following arrays */
    u_int8_t use_actual_nf; /* whether to use the
actual NF obtained or a hardcoded one */
    u_int32_t lteu_weight[LTEU_MAX_BINS]; /* weights for MU algo */
    u_int32_t lteu_thresh[LTEU_MAX_BINS]; /* thresholds for MU
algo */
    u_int32_t lteu_gamma[LTEU_MAX_BINS]; /* gamma's for MU algo */
    u_int32_t lteu_scan_timeout; /* timeout
in ms to gpio toggle */
    u_int32_t alpha_num_bssid; /* alpha for
num active bssid calculation */
    u_int32_t lteu_cfg_reserved_1; /* extra parameter added for
future use */
}

```

```

    } ieee80211req_lteu_cfg_t;

    struct ieee80211req_athdbg {
        u_int8_t cmd;
        u_int8_t dstmac[IEEE80211_ADDR_LEN];
        union {
            int param[4];
            ieee80211_rrm_beaconreq_info_t bcnrpt;
            ....
            ieee80211req_mu_scan_t mu_scan_req;
            ieee80211req_lteu_cfg_t lteu_cfg;
        } data;
    };
}

```

#### 9.4.3.2 Interaction between host software and target firmware interface

A new WMI command is added for host to configure MU related parameters run time. Firmware to use these parameters to determine the scan/MU start trigger and for MU computation .

```

WMI_SET_LTEU_CONFIG_CMDID
typedef struct {
    A_UINT32 gpio_enable;
    A_UINT32 num_lteu_bins;
    A_UINT32 mu_rssi_threshold[LTEU_MAX_BINS];
    A_UINT32 mu_weight[LTEU_MAX_BINS];
    A_UINT32 mu_gamma[LTEU_MAX_BINS];
    A_UINT32 mu_scan_timeout;
    A_UINT32 wifi_tx_power;
    A_UINT32 alpha_num_bssid;
    A_UINT32 use_actual_nf;
    A_UINT32 allow_err_packets;
} wmi_set_lteu_config;

```

#### 9.4.3.3 Wifitool commands and options

The wifitool can be used to invoke the IOCTLs for testing. It allows issuing MU and configuration commands.

- For configuration – **wifitool athX lteu\_cfg [ -g 0|1 ] [ -n 1-3 ] [ -w 1-3 0-100+ ] [ -y 1-3 0-100+ ] [ -t 1-3 (-110)-0+ ] [ -o 10-50 ] [ -h ]**
  - -g : gpio start or not
  - -n : number of bins

- w : number of weights followed by the individual weights
- y : number of gammas followed by individual gammas
- t : number of thresholds followed by individual thresholds
- o : timeout
- h : help
- For MU –**wifitool athX mu\_scan [ -i 1-99 ] [ -c 36-165 ] [ -t 1-15 ] [ -d 0-5000 ] [ -p 0-100 ] [ -b (-110)-0 ] [ -s (-110)-0 ] [ -u (-110)-0 ] [ -m 00000-999999 ] [ -a 0-100 ] [ -w ] [ -h ]**
  - i : request id
  - c : IEEE number of the channel
  - t : algo(s) to use
  - d : time
  - p : LTEu Tx power
  - b : RSSI Threshold for BSSID
  - s : RSSI Threshold for STA
  - u : RSSI Threshold for WCUBS detection
  - m : PLMN ID to distinguish between same and different operator WCUBS
  - a : alpha for num active bssid calc
  - w : wait for wireless event
  - h : help
- If the -w option is specified for MU, after issuing the ioctl, the tool waits for the associated wireless custom event IWEVGENIE and prints the values returned in the event.

#### MU report event:

**req\_id=val1** : The MU request id

**channel=val2** : The channel on which MU was done

**status=val3** : The MU status

**num\_bssid=val4** : Number of active BSSIDs

**actual\_duration=val5** : Duration for which MU was done

**total\_val[0]=val6** : Total MU calculated by basic binning algo

**total\_val[1]=val7** : Total MU calculated by enhanced binning algo

**total\_val[2]=val8** : Total MU calculated by hidden node algo

**mu\_hidden\_node=val9 val10 val11** : MU calculated by hidden node algo reported on a per bin basis

**num\_ta\_entries=val12** : Number of TA entries

The following prints are displayed only when num\_ta\_entries value is greater than zero.

**TA\_MU\_entry[1]= device\_type=val13 BSSID=val14 TA\_mac\_address=val15 Average\_duration=val16 Average\_RSSI=val17 MU\_percent=val18** : Entry 1 in TA based MU report in respective order: [device\_type, BSSID, TA mac address, Average duration, Average RSSI, MU%]

(The aforementioned entries are repeated for all TA entries in the MU report database)

**TA\_MU\_entry[2]=[....]**

[....]

[....]

**TA\_MU\_entry[N]=[....]**

#### 9.4.3.4 Sample configuration scenario

This sample configuration is considered to be performed in shielded environment for proper results.

To configure Wi-Fi for LTE-U and verify the working of this functionality, perform the following steps:

1. Bring up AP DUT with mod param lteu\_cfg=0. Initial print in FW is displayed to show LTE-U support is disabled. Any of the LTE-U-related commands do not work in this case and corresponding error messages are displayed.
2. Bring up AP DUT with mod param lteu\_cfg=1. Initial print in FW is displayed to show LTE-U support is enabled.
3. Set different values in lteu config using wifitool on AP DUT for no. of bins, weights, RSSI thresholds and gammas. Corresponding prints in FW are set for proper no. of bins, weights, RSSI thresholds and gammas.
4. Start MU computation on channel 40, where 36 is current channel of AP DUT. Channel switch from 36 to 40 occurs. MU computation start, end and report prints with correct values are displayed for channel 40.
5. Start MU computation on channel 36, where 36 is current channel of AP DUT. MU computation start, end and report prints with correct values are displayed for channel 36.
6. Start MU with different time durations on AP DUT. The MU end time minus start time displayed in the prints matches with the actual mu duration requested. If duration requested is 0, invalid MU is reported.
7. Start MU with different algorithm bit masks on AP DUT. The computed MU using all algorithms requested in the bit mask is reported. Only algorithm 1 is supported.
8. Bring up AP DUT on channel 36, HT80, 3x3 chain. Run (HT20, 1x1), (HT40, 1x1), (VHT80, 1x1), (HT20, 2x2), (HT40, 2x2), (VHT80, 2x2), (HT20, 3x3), (HT40, 3x3), (VHT80, 3x3) traffic between reference AP and reference STA on channel 36. AP DUT must be able to capture the packets and decode the preamble, compute and report MU using algorithm 1. Check the per packet RSSI and frame duration by enabling per packet log.
9. Bring up AP DUT on DFS channel 60, HT80, 3x3 chain. Run (HT20, 1x1), (HT40, 1x1), (VHT80, 1x1), (HT20, 2x2), (HT40, 2x2), (VHT80, 2x2), (HT20, 3x3), (HT40, 3x3), (VHT80, 3x3) traffic between reference AP and reference STA on channel 60. AP DUT must

- be able to capture the packets and decode the preamble, compute and report MU using algorithm 1. Check the per-packet RSSI and frame duration by enabling per packet log.
10. Bring up AP DUT on channel 36. Run 100 Mbps UDP traffic between ref AP and ref sta on channel 36 for MU duration=50ms, 100ms, 200ms. The computed MU must approximately be the same in all cases.
  11. Start MU from for 2 secs, and before the MU computation is completed, enter the MU start command again. The command must fail when it is entered for the second time, and must report one request is already active.

## 9.4.4 Display statistics and packet logs

### 9.4.4.1 Tx and Rx firmware statistics

The txrx\_fw\_stats command needs one input argument. The most commonly used argument options available and their use are listed as follows:

```
iwpriv ath<if> txrx_fw_stats <argument>
```

| Argument Option | Output                     |
|-----------------|----------------------------|
| 1               | Physical device statistics |
| 3               | Rx rate statistics         |

#### Physical device statistics

The physical device target data stats show the number of times various expected and unexpected transmit and receive events have happened. In addition to that it also displays remaining IRAM and DRAM memory in bytes.

The following is a sample output of the iwpriv ath0 txrx\_fw\_stats 1 command:

```
iwpriv ath0 txrx_fw_stats 1
WAL Pdev stats:

### Tx ####
comp_queued      :    7
comp_delivered   :    7
msdu_enqueued    : 196
wmm_drop          :    0
local_enqueued   : 196
local_freed       : 196
hw_queued         : 196
hw_reaped         : 196
mac underrun      :    0
phy underrun      :    0
```

```

tx_abort : 0
mpdus_requeued : 0
excess_retries : 1
last_rc : 3
sched_self_trig : 0
ampdu_retry_failed: 0
illegal_rate_errs : 0
pdev_cont_xretry : 0
pdev_tx_timeout : 0
pdev_resets : 1
ppdu_txop_ovf : 0
mcast_Drop : 0

### Rx ####
ppdu_route_change : 0
status_rcvd : 21
r0_frags : 0
r1_frags : 0
r2_frags : 0
r3_frags : 0
htt_ms dus : 21
htt_mpdus : 21
loc_ms dus : 17
loc_mpdus : 17
oversize_amsdu : 0
phy_errs : 0
phy_errs_dropped : 0
mpdu_errs : 0

##### Free memory#####

IRAM_Remaining: 6308
DRAM_Remaining: 3768

```

The following list describes the fields displayed in the Rx section of the output:

- ppdu\_route\_change: # of times for a received PPDU, part of MPDUs are data frames and part of the MPDUs are non-data frames
- status\_rcvd: # of Rx status is used. One Rx status usually represents one MSDU
- r0\_frags: # of buffer fragmentation happened in Ring 0. The buffer fragmentation means that a MSDU occupies more than one Rx buffer

- r1\_frags: # of buffer fragmentation happened in Ring 1
- r2\_frags: # of buffer fragmentation happened in Ring 2
- r3\_frags: # of buffer fragmentation happened in Ring 3
- htt\_msdu: # of data MSDUs received
- htt\_mpdu: # of data MPDUs received
- loc\_msdu: # of non-data MSDUs received
- loc\_mpdu: # of non-data MPDUs received
- oversize\_amsdu: # of the times that receiving an A-MSDU which has SDUs more than the size of Rx status ring
- phy\_errs : # of received PPDUs as Phy errors
- phy\_errs\_dropped : # of received PPDUs as Phy errors dropped
- mpdu\_errs :
- # of MPDU received with error.

### Receive rate control statistics

The Rx rate statistical details display fields, such as the number of times Rx frames were received using different rates modulation, coding, short vs. long GI, and RSSI.

The following is a sample output of the iwpriv ath0 txrx\_fw\_stats 3 command:

```
iwpriv ath0 txrx_fw_stats 3
RX Rate Info:
MCS counts (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
SGI counts (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
NSS counts: 1x1 0, 2x2 0, 3x3 0, 4x4 0
NSTS count: 0
BW counts: 20MHz 0, 40MHz 0, 80MHz 0
Preamble counts: 0, 0, 0, 0, 0, 0
STBC rate counts (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
LDPC TXBF Counts: 0, 0
RSSI (data, mgmt): 0, 13
RSSI Chain 0 (0x00 0x00 0x00 0x00)
RSSI Chain 1 (0x00 0x00 0x00 0x00)
RSSI Chain 2 (0x00 0x00 0x00 0x00)
```

The following list describes the output fields:

- MCS Counts: these are counters for each MCSs 0..9 in case of VHT, and MCS0..7 in the case of the HT association. For 802.11n MCS8..23 please combine this field with NSS field, e.g, MCS8 is NSS 2 MCS0. Please note that this does not capture legacy OFDM/CCK rates.
- SGI counts: counters for each SGI enabled MCS

- NSS counts: Captures Number of Spatial Streams. Indicate whether 1x1, 2x2 or 3x3 rate is being used. Combined with MCS gives actual (802.11n) MCS in case of HT.
- NSTS count: Indicates whether the frames are being sent with STBC enabled and the transmission is at a 1x1 rate. The NSTS count can be seen to be equal to the sum of STBC counts.
- BW counts: indicate number of received frames on 20, 40 and 80MHz. Useful to debug which all BWs are being used currently by the transmitter STA.
- Preamble counts: index 0 counts legacy (CCK/OFDM) ppdu, 1 HT, 2 HT with BF (on QCA9880 always 0), 3 VHT and 4 VHT with BF (on QCA9880 always zero), 5 all other, e.g., phy error
- STBC rate counts: Similar to MCS counts give what all MCSs have STBC enabled.
- LDPC TXBF counts: the first counter increments for each received LDPC ppdu. Second one increment for each received TxBF frames, which is not supported by QCA9880.
- RSSI (data, mgmt.): absolute RSSI value as seen in received MAC descriptor for data and management frame respectively
- RSSI Chain 0/1/2: (sec80, sec40, sec20, pri20) gives rssi seen in MAC descriptor for given chain across primary/secondary channels. This could be quite useful to make sure, all chains are balanced.

**NOTE** Wen rate is fixed, only one of MCS counts is incremented with autorate. Most of the MCSs are used depending on the environment.

#### 9.4.4.2 Packet log analysis

Pktlog tool captures the TX, RX descriptors in real time and provides rates statistics, RSSI statistics, aggregate information, data type description, RA/TA/ BSSID addresses, MPDU sequence numbers, BA, TID, MSDU length and timestamp.

Pktlog tool essentially consists of `ath_pktlog.ko` (driver), `pktlogconf` (user space utility) and the `pktlogdecoder_11ac.pl` perl script.

Assuming that the pktlog is compiled into the code base, perform the following procedures to successfully set up the pktlog for QCA9880:

##### Collect pktlog and analyze pktlog data

To collect pktlog, use the `pktlogconf` command.

```
root@OpenWrt:/# pktlogconf -h
```

Usage of the packet log configuration command is as follows

Usage: `pktlogconf [-a adapter] [-e[event-list]] [-d adapter] [-s log-size] [-t -k -l]`

`[-b -p -i]`

- h show this usage
- a configures packet logging for specific 'adapter';  
configures system-wide logging if this option is  
not specified
- d disable packet logging
- e enable logging events listed in the 'event-list'  
event-list is an optional comma separated list of one or more  
of the following: rx tx rcf rcu ani (eg., pktlogconf -erx,rcu,tx)
- s change the size of log-buffer to "log-size" bytes
- t enable logging of TCP headers
- k enable triggered stop by a threshold number of TCP SACK packets
- l change the number of packets to log after triggered stop
- b enable triggered stop by a throughput threshold
- p enable triggered stop by a PER threshold
- i change the time period of counting throughput/PER

To analyze pktlog use, pktlogdecoder\_11ac.pl command.

`pktlogdecoder_11ac.pl`

Usage: `./pktlogdecoder_11ac.pl`

```
[ -agmnprtuvBDERGKJISVP] [-d <delta>] [-D <start>,<end>]
[-T <seqStart>,<ackStart>] [-I <offset>,<slot>]
(-x | -X ap_addr | pktLog.dat)
```

The following are the basic options:

- -h Print this message

The following are the options for the record-by-record log display (default):

- -a show all
- -v show detailed EVM info in log

The following are the options for displaying info other than the record-by-record log (each of the following is mutually-exclusive):

- -D show full Tx desc info from <start index> to <end index>
- -G print A-MPDU stats
- -R print rate statistics

- -K print PHY error stats
- -S print RSSI statistics
- -V print EVM statistics

### Enable packet log and view statistics

To enable pktlogconf on wifi1, enter the following command:

```
pktlogconf -a wifi1 -e
```

To stop pktlog, enter the following command:

```
pktlogconf -d wifi1
```

After pktlog is stopped, the pktlog can be copied from the following sysfile:

```
/proc/ath_pktlog/wifi1
```

Example: cp /proc/ath\_pktlog/wifi1 /tmp/pktlog.dat

To perform a file transfer using TFTP, use following commands:

- For AP135, “tftp <ip>”
- For IPQ4019, “tftp-hpa -m binary <ip>”

IPQ4019 tftp-hpa tool, by default, uses ASCII mode to transfer file. If –m binary is not used, the transferred pktlog file is corrupt if transferred in ASCII mode.

To display rate statistics:

```
pktlogdecoder_11ac.pl -R pktlog-wifi1-AP135-10.2.2.18_default.dat
File endianness: Big (AP)
File version: 10010
File size: 1048544 bytes
MAC Ver/Rev: 0x0/0x0 (Owl 1.0 - sw hardcoded version)
Number of records: 4920
Finished loading file

-- Tx Rate Statistics --

      Num      Avg #          Subframes Failures    PER      Excess Failures
Excess Failures %           %          Good     Bad
Rate   Frames  Subfr       %          RTS      Data
RTS     Data
-----
-----
vM09x3h    537    10.0 100.00%      5358     0      0.00%      0      0
0.00%    0.00%
-----
-----
Total Data frames : 537
Total MSDUs       : 16069
Total RTS failures : 0 (PER = 0.00%)
Total Data failures : 0 (PER = 0.00%)
Note: Excessive Retries (ERs) are accounted for by the last rate of the series.
```

```
-- Rx Rate Statistics --

      Num Avg #          Subframes          Weighted          ACK/BA
      Rate  Frames Subfr    %   Good   Bad    PER    PER Retries Fails <--- %
-----vM09x3h  40  18.9 100.00%    757     0   0.00%  0.00%       0     0   0.00%
-----
Total Data frames : 40
Total MSDUs       : 2226
Total A-MPDU Subframe PER : 0.00% (Good=757, Bad=0)
Total retries      : 0 (0.00% of good frames)
ACK/BA (dup seq) failures : 0.00%
```

#### 9.4.5 Known limitations

- If the AP DUT is brought up in HT20 mode, it wont be able to decode the MAC header of HT40/ VHT80 frames. It will only be able to decode the preamble. In that case, such frames are listed under “Unknown BSSID” category for MU computation per BSSID.
- If the AP DUT is brought up in HT40 mode, it wont be able to decode the MAC header of VHT80 frames. It will only be able to decode the preamble. In that case, such frames are listed under “Unknown BSSID” category for MU computation per BSSID.

### 9.5 TxOPs of 10 ms for best effort access category

According to ETSI regulatory domain rules, for a Best Effort (BE) access category AP in 5 GHz band, the transmit opportunities (TxOPs) that are greater than 6ms in duration must enable double backoff. To comply with the ETSI regulatory domain rules, TxOP can be extended to a maximum value of 10ms for BE access category in AP mode in 5GHz band. This TxOP of 10ms is supported only if the random back off (CW) values, which follow the TxOPs with longer duration, are doubled.

Due to doubling rule, the 10ms AP has less number of TxOPs, but a longer duration for each TxOP that causes an effective airtime.

To enable the double backoff support, enter the wifitool ath0 beeline\_fw\_test 155 1 command. Alternatively, to disable the double backoff support, enter the wifitool ath0 beeline\_fw\_test 155 0 command.

Prior to the implementation of this feature, each access category was preprogrammed with fixed CWmin and CWmax values and the TxOP's is within <=8ms. HW/MAC generated a random backoff (CW) within CWmin and CWmax values and transmission was performed for TxOP. This resulted in less number of TxOPs with a longer duration for each TxOP.

ETSI priority class is the EDCA access category (AC). Priority class 4 corresponds to AC\_VO, priority class 3 corresponds to AC\_VI, the priority class 2 corresponds to AC\_BE, and the priority class 1 corresponds to AC\_BK.

### 9.5.1 Configure double backoff for 10 ms TxOPs

A new FWTEST command is added to enable or disable the ETSI double backoff for 10 ms TxOP support. By default, per the ETSI regulatory support, Cwmin and Cwmax values for best effort AC is programmed to 15 and 63. ETSI parameters values for AP and STA are referred above. This results in random backoff (CW) generated is within 3 to 5.

To double the CW value, Cwmin can have either 15/31 and Cwmax can have either 31/63 for some duty cycle. In the algorithm, FW maintains the Avg TxOP for the Best Effort (BE) access category using  $Tav = 0.9*Tav + 0.1* \text{current TxOP}$ .

If Avg TxOP  $Tav$  exceeds 6ms, FW calculates the duty cycle for Tcwmin15, Tcwmin31, Tcwmax31, and Tcwmax63 every 100ms using the following formula:

- CWmin is duty cycled between 15 and 31, every 100 ms
  - $Tcwmin31 = 100*(Tav - 6)/4$
  - Tcwmin31 varies between 0 and 100 ms (for  $Tav$  between 6 and 10 ms)
  - $Tcwmin15 = 100 - Tcwmin31$
  - Tcwmin15 varies between 100 and 0 ms
- CWmax is duty cycled between 63 and 31, every 100 ms
  - $Tcwmax31 = 0.8*Tcwmin31$
  - Tcwmax31 varies between 0 and 80 ms (80% of Tcwmin31)
  - $Tcwmax63 = 100 - Tcwmax31$
  - Tcwmax63 varies between 100 and 20 ms

The Cwmin and Cwmax for the BE queue is programmed for the calculated duty cycle whereby the CW value is doubled to meet the requirement. When the  $Tav < 6\text{ms}$ , the default value of (Cwmin,Cwmax) of (15,63) is programmed.

For example, when average TxOP duration is calculated to be 8 ms ( $Tav = 8\text{ ms}$ ), the (Cwmin and Cwmax) programmed is as follows:

- CWmin is duty cycled between 15 and 31, every 100 ms
  - $Tcwmin31 = 100*(8 - 6)/4 = 50\text{ ms}$   
Tcwmin31 varies between 0 and 100 ms (for  $Tav$  between 6 and 10 ms)
  - $Tcwmin15 = 100 - Tcwmin31 = 50\text{ ms}$   
Tcwmin15 varies between 100 and 0 ms
- CWmax is duty cycled between 63 and 31, every 100 ms
  - $Tcwmax31 = 0.8*Tcwmin31 = 40\text{ ms}$   
Tcwmax31 varies between 0 and 80 ms (80% of Tcwmin31)
  - $Tcwmax63 = 100 - Tcwmax31 = 60\text{ ms}$   
Tcwmax63 varies between 100 and 20 ms

In a duration of 100ms, (CWmin , CWmax) is changed as (31, 31) for the first 40ms, then to (31,63) for the subsequent 10ms, and finally to (15,63) for the remaining 50ms. This type of setting implements the doubling CW requirement.

### **Tx sequence completion ISR**

In the Tx sequence completion interrupt service routine (ISR), the last TxOP time for the transmitted packet is accumulated until the burst sequence is completed. The average TxOP is computed and stored in the wal\_txq structure.

$$\text{Tav} = 0.9 * \text{Tav} + 0.1 * \text{current TxOP}$$

### **Tx send post-PPDU**

In Tx send module, before PPDU Posting , the Avg TxOP in the Best Effort txq is checked and if it is > 6ms, the duty cycle for CWmin15, CWmin31 , CWmax31 and CWmax63 is computed as per the System team Suggesting. This computation is done for every 100ms interval and CWmin, CWmax value for the Best Effort TxQ is changed accordingly to the duty cycle.

When the average TxOP is less than 6ms, the default CW (min,max) of (15,63) is restored.

### **WMI control path changes**

In the WMI command path, changes are added to detect the ETSI reg\_domain in the WMI\_REGDOMAIN\_CMDID and store it in \_wal\_pdev structure for further use in the data path. Because this functionality to support TxOP of 10 milliseconds with double backoff support is specific to ETSI domain and for 5 GHz, checks are added in the data path (tx\_seq and tx\_send module) to enable the support for ETSI domain only. In addition, FW\_TEST\_CMD is also provided to enable or disable the feature.

## **9.5.2 Sample configuration scenario**

Check the TxOPs stats using the wifitool ath0 beeline\_fw\_test 154 0 command. Set the country to Germany to configure ETSI regulatory domain using either of the following commands

```
iwpriv wifi0 setCountry 'DE'
```

or

```
uci set wireless.@wifi-device[0].country=0x114
```

The following is a sample AP configuration:

```
config wifi-device 'wifi0'
    option type 'qcawifi'
    option channel '149'
    option htmode 'HT80'
    option rxchainmask '15'
    option txchainmask '15'
    option macaddr '8c:fd:f0:07:24:aa'
    option hwmode '11ac'
    option disabled '0'
    option country '0x114'

config wifi-iface
```

```

        option device 'wifi0'
        option network 'lan'
        option mode 'ap'
        option ssid 'public'
        option encryption 'none'

    config wifi-device 'wifi1'
        option type 'qcawifi'
        option channel 'auto'
        option macaddr '00:03:7f:86:00:34'
        option hwmode '11ac'
        option disabled '1'

    config wifi-iface
        option device 'wifi1'
        option network 'lan'
        option mode 'ap'
        option ssid 'OpenWrt'
        option encryption 'none'

    config wifi-device 'wifi2'
        option type 'qcawifi'
        option channel 'auto'
        option macaddr '8c:fd:f0:07:2e:42'
        option hwmode '11ng'
        option disabled '1'

    config wifi-iface
        option device 'wifi2'
        option network 'lan'
        option mode 'ap'
        option ssid 'OpenWrt'
        option encryption 'none'

```

By default, the regdomain is set to U.S. For default regdomain, TxOPs values should be zero.

```

root@OpenWrt:/# wifitool ath0 beeline_firmware_txop 154 0
[71986.792996] FIRMWARE:1ms Burst
[71986.792996] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.801102] FIRMWARE:txop_on_burst_complete = 0
[71986.801102]
[71986.807164] FIRMWARE:2ms Burst
[71986.807164] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
root@OpenWrt:/# [71986.816191] FIRMWARE:txop_on_burst_complete = 0
[71986.816191]
[71986.823589] FIRMWARE:3ms Burst
[71986.823589] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.832684] FIRMWARE:txop_on_burst_complete = 0
[71986.832684]
[71986.838745] FIRMWARE:4ms Burst
[71986.838745] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.847776] FIRMWARE:txop_on_burst_complete = 0
[71986.847776]
[71986.853798] FIRMWARE:5ms Burst
[71986.853798] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.862895] FIRMWARE:txop_on_burst_complete = 0
[71986.862895]

```

```
[71986.868956] FIRMWARE:6ms Burst
[71986.868956] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.877983] FIRMWARE:txop_on_burst_complete = 0
[71986.877983]
[71986.884007] FIRMWARE:7ms Burst
[71986.884007] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.893104] FIRMWARE:txop_on_burst_complete = 0
[71986.893104]
[71986.899164] FIRMWARE:8ms Burst
[71986.899164] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.908192] FIRMWARE:txop_on_burst_complete = 0
[71986.908192]
[71986.914217] FIRMWARE:9ms Burst
[71986.914217] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.923316] FIRMWARE:txop_on_burst_complete = 0
[71986.929374] FIRMWARE:10ms Burst
[71986.929374] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.938495] FIRMWARE:txop_on_burst_complete = 0
[71986.944510] FIRMWARE:11ms Burst
[71986.944510] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.953679] FIRMWARE:txop_on_burst_complete = 0
[71986.953679]
[71986.959771] FIRMWARE:12ms Burst
[71986.959771] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.968891] FIRMWARE:txop_on_burst_complete = 0
[71986.968891]
[71986.974892] FIRMWARE:13ms Burst
[71986.974892] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.984076] FIRMWARE:txop_on_burst_complete = 0
[71986.984076]
[71986.990136] FIRMWARE:14ms Burst
[71986.990136] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71986.999257] FIRMWARE:txop_on_burst_complete = 0
[71986.999257]
[71987.005274] FIRMWARE:15ms Burst
[71987.005274] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[71987.014440] FIRMWARE:txop_on_burst_complete = 0
[71987.014440]
[71987.020532] FIRMWARE:Bkoff Config cnt: g_63_15_cnt = 0, g_31_31_cnt =
0
[71987.020532]
[71987.028720] FIRMWARE:Bkoff Config cnt: g_max_15_cnt = 0, g_63_min_cnt =
0 , Total_Hw_chg_cnt = 0
```

After changing to ETSI Domain, for the TxOPs occurring after 6 ms burst, they must be double backed-off as displayed in statistics.

```
root@OpenWrt:/# wifitool ath0 beeline_firmware_test 154 0
[67256.052989] FIRMWARE:1ms Burst
[67256.052989] avg_txop_burst_complete = 20701 ,avg_txop_on_100ms = 151
[67256.061612] FIRMWARE:txop_on_burst_complete = 16717
[67256.061612]
[67256.068045] FIRMWARE:2ms Burst
[67256.068045] avg_txop_burst_complete = 3734 ,avg_txop_on_100ms = 92
[67256.077384] FIRMWARE:txop_on_burst_complete = 8280
[67256.077384]
```

```
[67256.083669] FIRMWARE:3ms Burst
[67256.083669] avg_txop_burst_complete = 2576 ,avg_txop_on_100ms = 58
[67256.093098] FIRMWARE:txop_on_burst_complete = 220
[67256.093098]
[67256.099349] FIRMWARE:4ms Burst
[67256.099349] avg_txop_burst_complete = 3782 ,avg_txop_on_100ms = 142
[67256.108815] FIRMWARE:txop_on_burst_complete = 5828
[67256.108815]
[67256.115091] FIRMWARE:5ms Burst
[67256.115091] avg_txop_burst_complete = 507 ,avg_txop_on_100ms = 26
[67256.124435] FIRMWARE:txop_on_burst_complete = 231
[67256.124435]
[67256.130682] FIRMWARE:6ms Burst
[67256.130682] avg_txop_burst_complete = 310 ,avg_txop_on_100ms = 20
root@OpenWrt:/# [67256.139991] FIRMWARE:txop_on_burst_complete = 637
[67256.139991]
[67256.147613] FIRMWARE:7ms Burst
[67256.147613] avg_txop_burst_complete = 580 ,avg_txop_on_100ms = 43
[67256.156924] FIRMWARE:txop_on_burst_complete = 48
[67256.156924]
[67256.163010] FIRMWARE:8ms Burst
[67256.163010] avg_txop_burst_complete = 4904 ,avg_txop_on_100ms = 388
[67256.172544] FIRMWARE:txop_on_burst_complete = 5042
[67256.172544]
[67256.178886] FIRMWARE:9ms Burst
[67256.178886] avg_txop_burst_complete = 26 ,avg_txop_on_100ms = 2
[67256.187976] FIRMWARE:txop_on_burst_complete = 107
[67256.187976]
[67256.194171] FIRMWARE:10ms Burst
[67256.194171] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[67256.203346] FIRMWARE:txop_on_burst_complete = 0
[67256.203346]
[67256.209439] FIRMWARE:11ms Burst
[67256.209439] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[67256.218528] FIRMWARE:txop_on_burst_complete = 0
[67256.218528]
[67256.224553] FIRMWARE:12ms Burst
[67256.224553] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[67256.233742] FIRMWARE:txop_on_burst_complete = 3
[67256.233742]
[67256.239804] FIRMWARE:13ms Burst
[67256.239804] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[67256.248924] FIRMWARE:txop_on_burst_complete = 0
[67256.248924]
[67256.254935] FIRMWARE:14ms Burst
[67256.254935] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[67256.264108] FIRMWARE:txop_on_burst_complete = 0
[67256.264108]
[67256.270201] FIRMWARE:15ms Burst
[67256.270201] avg_txop_burst_complete = 0 ,avg_txop_on_100ms = 0
[67256.279321] FIRMWARE:txop_on_burst_complete = 6
[67256.279321]
[67256.285318] FIRMWARE:Bkoff Config cnt: g_63_15_cnt = 488, g_31_31_cnt
= 434
[67256.285318]
```

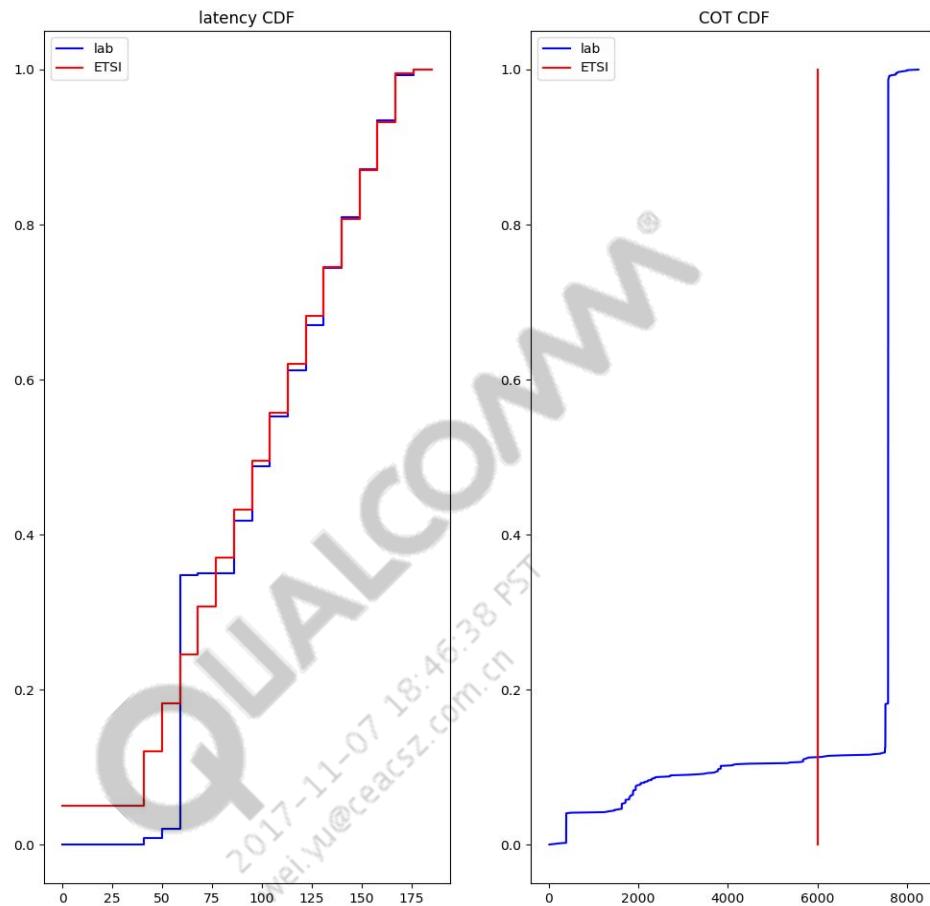
```
[67256.293882] FIRMWARE:Bkoff Config cnt: g_max_15_cnt = 433, g_63_min_
cnt = 433 , Total_Hw_chg_cnt = 1300
```

## 9.6 ETSI compliance for COT and CAT on QCA9980 and QCA9880

QCA9980 family boards (QCA9980, QCA9984, IPQ4019, and QCA9886) and QCA9880 platforms had failed some of the European Telecommunications Standards Institute (ETSI) regulation test cases. These cases were related to Channel Occupancy Test (COT) and Channel Access Latency (CAL) tests. An enhancement to address the failures and enable the ETSI regulatory compliance is implemented. The following sections explain the failures in detail and the design changes to address them.

### 9.6.1 CAL and COT considerations

This section describes the considerations for Channel Access Latency (CAL) tests and Channel Occupancy Tests (CAT).



**Figure 9-1 Failure Case CAT and COT**

To pass the COT, the following design points to be considered and implemented:

- For BE AC, SIFS burst of only 6 msec to be allowed for SU traffic.
- Not a single SIFS burst to exceed 6 msec.
- MU traffic to allow SIFS burst till 10 msec. Double backoff algorithm to be implemented for MU traffic cases. This technique ensures the proper gain that MU requires.
- Burst duration to be configurable per AC.

To pass the CAL test, the normal backoff latency must not exceed the ETSI regulations. To achieve this behavior, the burst duration and AIFSN values are to be configurable per AC.

## 9.6.2 Design methodology

The approach of FES duration TLV per PPDU is used to address this problem.

The existing double backoff algorithm is to be used for MU traffic. To achieve this implementation, the following design points are considered:

1. Timestamp stored per txq. It is reset to 0xFFFFFFFF at the start of MU traffic identified by n\_users > 1. A flag in txq is SET to identify the MU traffic.
2. The double backoff algorithm is enabled to configure the HW.
3. As soon as MU traffic ends – n\_users = 1, the timestamp stores the current value and the txq flag is reset.
4. If the time exceeds the default threshold limit of 2 seconds, without MU traffic, then the double backoff algorithm is disabled and the SU traffic burst limits restored.
5. The threshold limit is to be made configurable.

## 9.6.3 Test methods

To perform unit testing, the following methods are used:

- Debug logs to be used.
- Double backoff algorithm implementation had TX\_DEBUG\_STATS identifying burst duration array, which denotes whether burst duration is exceeded or not. This is to be used.

To perform integration testing, the following methods are used:

- Cabled setup – PvT lite setup can be used to test throughput cases for all boards – QCA9980, QCA9984, IPQ4019, and QCA9886 with all 11a, 11na, 11ng, 11ac combinations.
- Over-the-air (OTA) testing to be done and burst duration limits to be ensured.

To perform testing over ETSI setup.

1. ETSI testbed should be made available for continual testing throughout the development as the code changes are solely dependent on the results of the setup.
2. ETSI testbed need to be ready for MU testing also.

## 9.6.4 Limitations

The throughput tests might be done, and a minimal impact might be observed when ETSI tests are enabled (not in the normal scenarios).

## 9.7 5.9 GHz frequency band

Support for the 5.9 GHz band is introduced. Customers require 5 GHz radio to support up to 5.875 GHz for India opportunities, and also for other countries where the same band as the band in India is open and can be utilized. This new regulatory domain code can fulfill regulatory requirements

and does not impact existing domain code support. For 2.4 GHz, 13 channels are available. The 5 GHz channelization list for the regulatory domain code is as follows:

- 5150-5350 MHz
- 5725-5850 MHz
- 5825-5875 MHz for 20 MHz channel only

| BW  | Channel list   |
|-----|--|
| 20  | 5180, 5200, 5220, 5240, 5260, 5280, 5300, 5320, 5745, 5765, 5785, 5805, 5825, 5845, 5865 |
| 40  | 5190, 5230, 5270, 5310, 5755, 5795   |
| 80  | 5210, 5290, 5775   |
| 160 | 5250   |

## 9.7.1 Design overview

The 5 GHz regulatory domain APL19 is added to support up to 5.875 GHz. A new country “INDIA2” to use APL19\_ETSIC is added. Both direct attach (DA) and offload (OL) architectures are supported.

### 9.7.1.1 Add 5G regdomain APL19 to support up to 5.875 GHz

For DA and OL:

```
static const REG_DOMAIN ahCmnRegDomains[] = {
    ...
    // add 5G regdomain APL19 to support up to 5.875GHz
    {APL19, FCC, NO_DFS, PSCAN_FCC_T | PSCAN_FCC, NO_REQ,
     BM(F13_5180_5240, F10_5260_5320, F4_5745_5825, F2_5845_5865, -1, -1,
     -1, -1, -1, -1, -1),
      CHAN_TURBO_G_BMZERO,
      CHAN_TURBO_G_BMZERO,
      BMZERO,
      BMZERO,
      CHAN_TURBO_G_BMZERO
    },
    ...
}
```

The maximum allowed antenna gain is same as F4\_5745\_5825.

```
static const REG_DMN_FREQ_BAND regDmn5GhzFreq[] = {
    ...
    // define F2_5845_5865
    { 5845, 5865, 30, 0, 20, 20, NO_DFS, NO_PSCAN, 0 }, /* F2_5845_5865 */
    ...
}
```

### 9.7.1.2 Add country “INDIA1” to use APL19\_ETSIC

For DA:

```
static const COUNTRY_CODE_TO_ENUM_RD ahCmnAllCountries[] = {
    ...
    // Add country "INDIA2" to use APL19_ETSIC
    {CTRY_INDIA2,          APL19_ETSIC,      "IN", "INDIA2",           YES,   NO,
     YES, YES, YES, YES, YES, 7000 },
    ...
}
```

For OL:

```
static const COUNTRY_CODE_TO_ENUM_RD ahCmnAllCountries[] = {
    ...
    // Add country "INDIA2" to use APL19_ETSIC
    {CTRY_INDIA2,          APL19_ETSIC,      "IN", "INDIA2",           YES,   NO,
     YES, YES, YES, YES, YES, YES, YES, YES, 7000 },
    ...
}
```

### 9.7.1.3 Define APL19\_ETSIC

For DA and OL:

```
enum EnumRd {
    ...
    APL19_ETSIC= 0x71,/* India with 5.9GHz support */
    ...
}
static const REG_DMN_PAIR_MAPPING ahCmnRegDomainPairs [] = {
    ...
    // Add APL19_ETSIC
    {APL19_ETSIC,      APL19,                 ETSIC,           NO_REQ, NO_REQ, PSCAN_
     DEFER, 0 },
    ...
}
```

### 9.7.1.4 Define APL19

```
enum EnumRd {
    ...
    APL19= 0x1240,/* India with 5.9GHz support */
    ...
}
```

### 9.7.1.5 Define CTRY\_INDIA1

For DA and OL:

```
enum CountryCode {
    ...
    CTRY_INDIA2= 5006,/* India with 5.9GHz support */
    ...
}
```

```
...
}
```

### 9.7.1.6 Limit BW 40/80/160 on high band

For DA:

```
bool __ahdecl
ath_hal_init_channels(struct ath_hal *ah,
                      HAL_CHANNEL *chans, u_int maxchans, u_int *nchans,
                      u_int8_t *regclassids, u_int maxregids, u_int *nregids,
                      HAL_CTRY_CODE cc, u_int32_t modeSelect,
                      bool enableOutdoor, bool enableExtendedChannels,
                      bool block_dfs_enable)
{
    ...
    for (cm = modes; cm < &modes[N(modes)]; cm++) {
        ...
        for (b=0;b<64*BMLEN; b++) {
            if (IS_BIT_SET(b,channelBM)) {

#ifndef ATH_NO_5G_SUPPORT
                if((b == F2_5845_5865) && IS_HT40_MODE(cm->mode) && (rd ==
&rd5GHz))
                    continue;
...
#endif
            }
        ...
    }
}
```

For OL:

```
bool __ahdecl
ol_regdmn_init_channels(struct ol_regdmn *ol_regdmn_handle,
                        struct ieee80211_channel *chans, u_int maxchans, u_int
*nchans,
                        u_int8_t *regclassids, u_int maxregids, u_int *nregids,
                        REGDMN_CTRY_CODE cc, u_int32_t modeSelect,
                        bool enableOutdoor, bool enableExtendedChannels,
                        bool block_dfs_enable)
{
    ...
    for (cm = modes; cm < &modes[N(modes)]; cm++) {
        ...
        /* Limit F2_5845_5865 only for BW20 */
        if ((cm->mode == REGDMN_MODE_11AC_VHT80_80) && (rd == &rd5GHz))
        {
            ol_regdmn_init_vht80_80_chan(&vht80_80_chans);

            if (regdmn != OVERRIDE_RD) {

```

```
// Find 80 M channels
for (b=0;b<64*BMLEN; b++) {
    if (IS_BIT_SET(b,channel1BM)) {
        if(b == F2_5845_5865)
            continue;
    ...
} else if ((cm->mode == REGDMN_MODE_11AC_VHT160) && (rd ==
&rd5GHz)) {

    ol_regdmn_init_vht80_80_chan(&vht160_chans);
    if (regdmn != OVERRIDE_RD) {
        // Find 160 M channels
        for (b=0;b<64*BMLEN; b++) {
            if (IS_BIT_SET(b,channel1BM)) {
                if(b == F2_5845_5865)
                    continue;
            ...
}
// Walk through the 5G band to find 80 Mhz channel
else if ((cm->mode == REGDMN_MODE_11AC_VHT80) && (rd == &rd5GHz))
{
    ol_regdmn_init_vht80_chan(&vht80_chans);

    if (regdmn != OVERRIDE_RD) {
        // Find 80 M channels
        for (b=0;b<64*BMLEN; b++) {
            if (IS_BIT_SET(b,channel1BM)) {
                if(b == F2_5845_5865)
                    continue;
            ...
}
...
for (b=0;b<64*BMLEN; b++) {
    if (IS_BIT_SET(b,channel1BM)) {
        if((b == F2_5845_5865) && (cm->mode != REGDMN_MODE_11NA_
HT20) && (cm->mode != REGDMN_MODE_11AC_VHT20) && (cm->mode != REGDMN_
MODE_11A))
            continue;
    ...
}
...
}
```

### **9.7.1.7 Add high channels to scan list**

For DA:

```
static const u_int16_t scan_order[] = {
    ...
    /* 6 FCC channel: 144, 149, 153, 157, 161, 165 */
    5720, 5745, 5765, 5785, 5805, 5825,
    /* Added India high channels 169, 173 */
    5845, 5865,
```

```
    ...
}
```

For OL:

```
static const u_int16_t default_scan_order[] = {
    ...
    /* 6 FCC channel: 144, 149, 153, 157, 161, 165 */
    5720, 5745, 5765, 5785, 5805, 5825,
    /* Added India high channels 169, 173 */
    5845, 5865,
    ...
}
```

## 9.7.2 Verify the 5.9 GHz band

### 9.7.2.1 Test procedure

- AP: AP152+QCA9886 STA: AP152+QCA9886
- AP: AP148+QCA9984 STA: AP148+QCA9984 (for HT160 test)

#### Configuration on AP

1. Enter UCI commands on console:

```
uci set wireless.wifi0.disabled=0
uci set wireless.@wifi-iface[0].mode=ap
uci set wireless.@wifi-iface[0].ssid=test-2g
uci set wireless.@wifi-iface[0].wds=1
uci set wireless.wifi0.country=5006
uci set wireless.wifil.disabled=0
uci set wireless.@wifi-iface[1].mode=ap
uci set wireless.@wifi-iface[1].ssid=test-5g
uci set wireless.@wifi-iface[1].wds=1
uci set wireless.wifil.country=5006
```

2. Bring up wifi0 and wifil on AP:

```
wifi up
```

3. View the channel list on AP to confirm it matches the requirement:

```
wlanconfig ath0 list channel
wlanconfig ath1 list channel
```

#### Configuration on STA

1. Enter UCI commands on console:

```
uci set wireless.wifi0.disabled=0
```

```

uci set wireless.@wifi-iface[0].mode=sta
uci set wireless.@wifi-iface[0].ssid=test-2g
uci set wireless.@wifi-iface[0].wds=1
uci set wireless.wifi0.country=5006
uci set wireless.wifi1.disabled=0
uci set wireless.@wifi-iface[1].mode=sta
uci set wireless.@wifi-iface[1].ssid=test-5g
uci set wireless.@wifi-iface[1].wds=1
uci set wireless.wifi1.country=5006

```

2. Bring up wifi0 and wifi1 on STA:

```
wifi up
```

3. Change channel and mode on AP.
4. Check the channel on STA. Each channel with desired mode must be OK to connect.
5. Perform a ping between AP and STA.
6. Repeat Steps 5 and 6 to complete all channel and mode combination test for both wifi0 and wifi1.

**NOTE** If wifi0 is 5 GHz radio and wifi1 is 2.4 GHz radio for AP148 + QCA9984, swap the SSID name.

### 9.7.2.2 DA test

APUT: AP152 Radio: wifi0

STA: AP152

Mode: 11ngh20 11ngh40

| Mode         | Channel list                              |
|--------------|---|
| 11ngh20      | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 |
| 11ngh40plus  | 1, 2, 3, 4, 5, 6, 7, 8, 9                 |
| 11ngh40minus | 5, 6, 7, 8, 9, 10, 11, 12, 13             |

### 9.7.2.3 OL test

#### Test on QCA9886

APUT: AP152 + QCA9886 Radio: wifi1

STA: AP152 + QCA9886

| Mode      | Channel list   |
|-----------|--|
| 11acvht20 | 5180, 5200, 5220, 5240, 5260, 5280, 5300, 5320, 5745, 5765, 5785, 5805, 5825, 5845, 5865 |
| 11acvht40 | 5190, 5230, 5270, 5310, 5755, 5795   |
| 11acvht80 | 5210, 5290, 5775   |

### Test on QCA9984 to verify VHT160 and OL 2.4G

APUT: AP148 + QCA9984 Radio: wifi1

STA: AP148 + QCA9984

| Mode         | Channel list                              |
|--------------|---|
| 11ngh20      | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 |
| 11ngh40plus  | 1, 2, 3, 4, 5, 6, 7, 8, 9                 |
| 11ngh40minus | 5, 6, 7, 8, 9, 10, 11, 12, 13             |

APUT: AP148 + QCA9984 Radio: wifi0

STA: AP148 + QCA9984

| Mode       | Channel list   |
|------------|--|
| 11acvht20  | 5180, 5200, 5220, 5240, 5260, 5280, 5300, 5320, 5745, 5765, 5785, 5805, 5825, 5845, 5865 |
| 11acvht40  | 5190, 5230, 5270, 5310, 5755, 5795   |
| 11acvht80  | 5210, 5290, 5775   |
| 11acvht160 | 5250   |

## 9.8 ETSI 5 GHz rules in Europe (ETSI EN 301 893)

### ETSI 5 GHz Channel Access Rules

ETSI is updating EN 301 893, which specifies 5 GHz channel access in Europe. The new rules are effective, starting June 2017, with the update requiring approval by the individual countries. The salient changes are as follows:

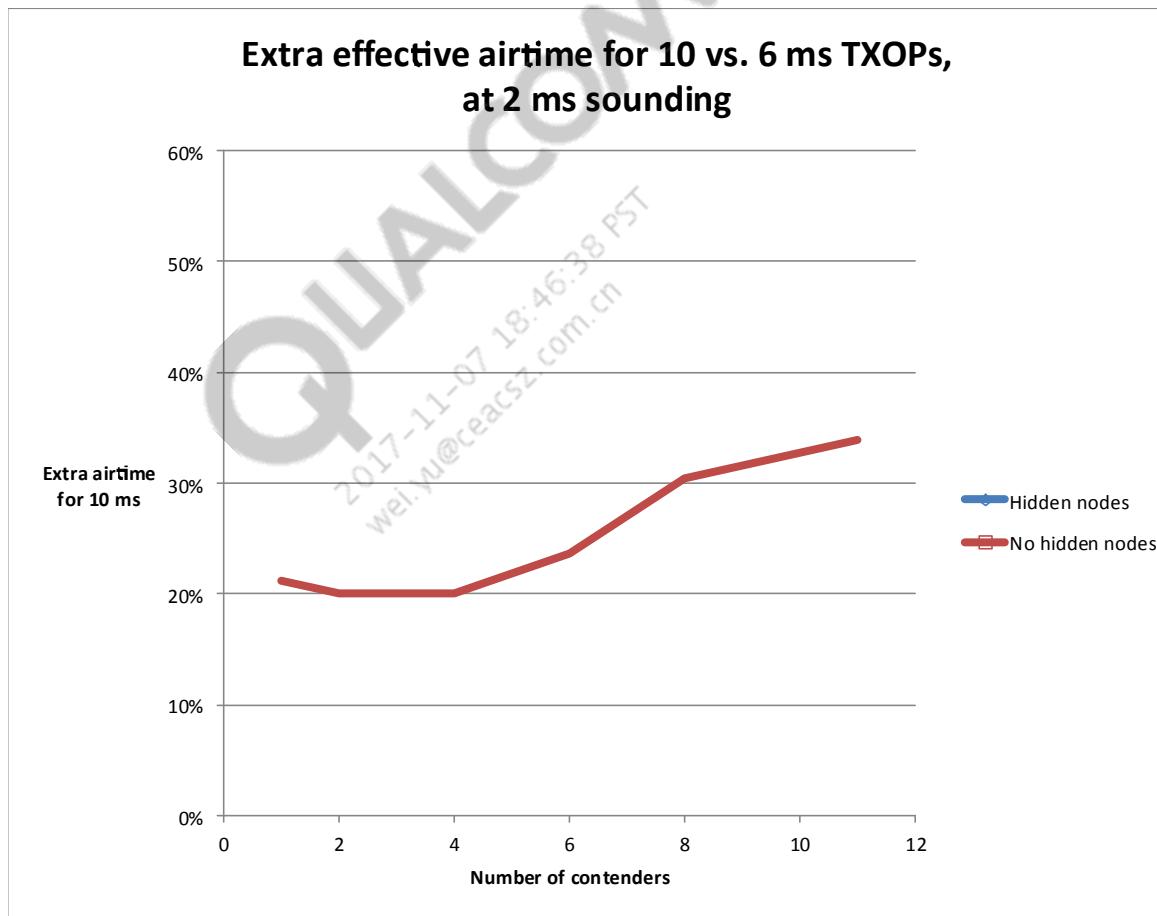
- A maximum TxOP limit of 6 ms, or 10 ms when CW doubling is applied
- -72 dBm ED for 11ax products; pre-11ax is allowed to continue to use -62 dBm ED + -82 dBm PD, although a similar waiver is likely also be requested for 11ax
- -72 dBm PD is allowed by ETSI

## 10 ms TxOPs in Europe

European regulations enable up to 10 ms TxOPs for APs operating in the 5 GHz band. However, TxOPs longer than 6 ms must be compensated using a double random range on the backoff that follows the long TxOP. The CW sequence effectively becomes 31/31/63 for an AP, instead of 15/31/63. This is referred to as *CW doubling*.

Because of CW doubling, a 10 ms AP has less but longer TxOPs. An AP with TxOPs of 10 ms has a more effective airtime than an AP with TxOPs of 6 ms, especially, when part of the TXOP is used for sounding. TxOPs of 10 ms are also beneficial without sounding, although in the presence of interference.

### Extra effective airtime for 10 ms vs. 6 ms TXOPs

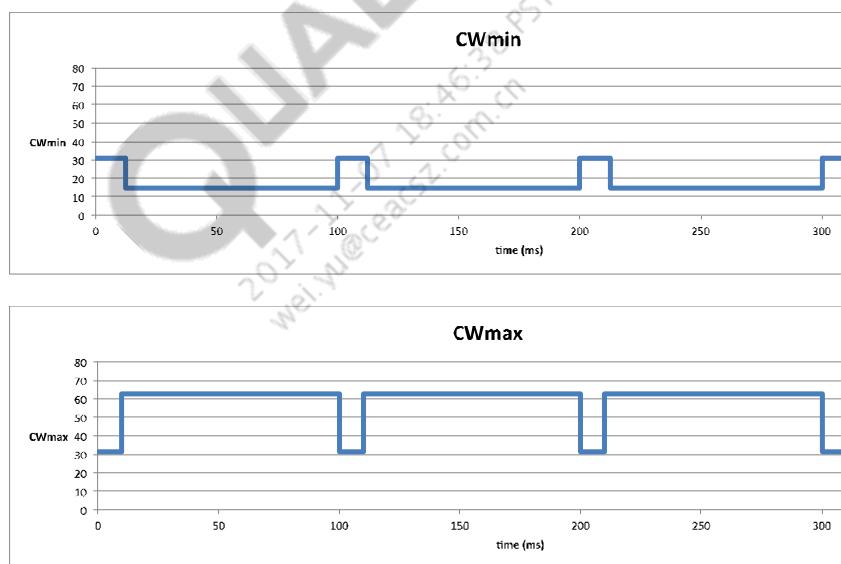


### Doubling CW implementation—Duty cycling

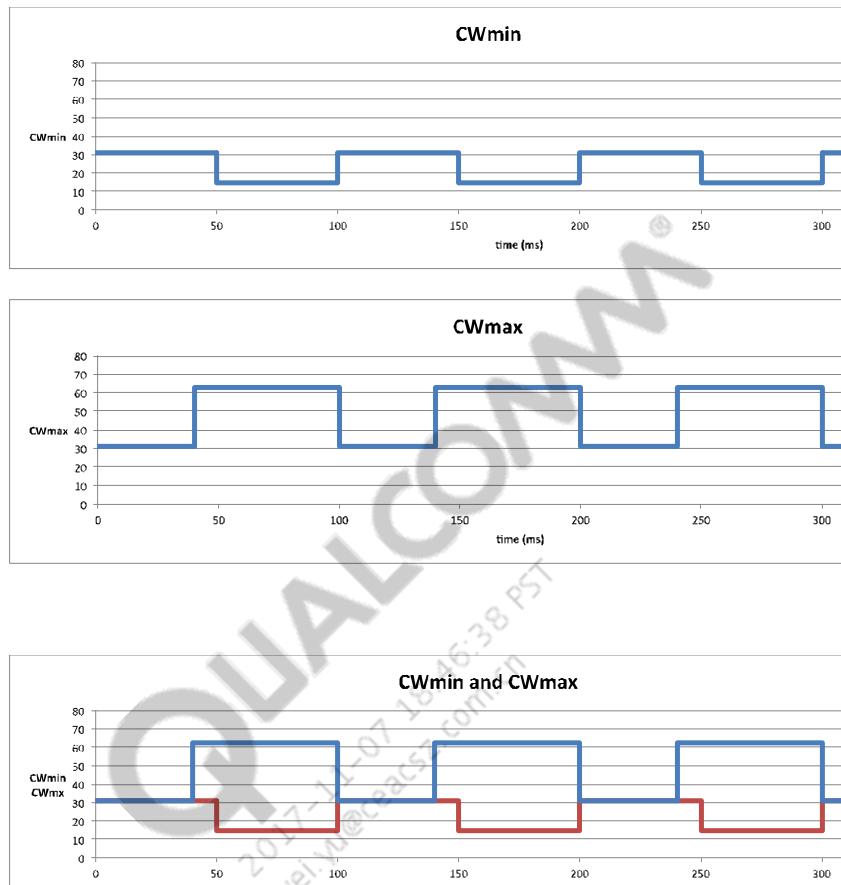
- TXOPs are generated as needed, with a duration of up to 10 ms
- SW maintains a moving average of the TXOP size, Tav
  - $Tav = 0.9*Tav + 0.1*TXOP$
- When the average TXOP size exceeds 6 ms (for example, if  $Tav > 6$  ms)

- CWmin is duty cycled between 15 and 31, every 100 ms
  - $Tcwmin31 = 100 * (Tav - 6) / 4$ 
    - $Tcwmin31$  varies between 0 and 100 ms (for  $Tav$  between 6 and 10 ms)
  - $Tcwmin15 = 100 - Tcwmin31$ 
    - $Tcwmin15$  varies between 100 and 0 ms
- CWmax is duty cycled between 63 and 31, every 100 ms
  - $Tcwmax31 = 0.8 * Tcwmin31$ 
    - $Tcwmax31$  varies between 0 and 80 ms (80% of  $Tcwmin31$ )
  - $Tcwmax63 = 100 - Tcwmax31$ 
    - $Tcwmax63$  varies between 100 and 20 ms
- Else (when  $Tav \leq 6$  ms), CWmin = 15 and CWmax = 63 (fixed)

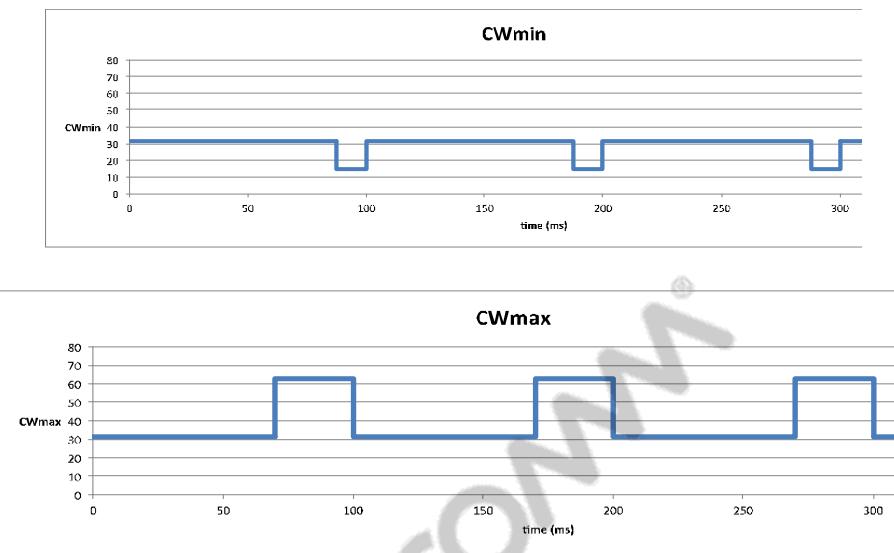
### Duty Cycling Example 1 - Average TXOP = 6.5 ms



## Duty Cycling Example 2 - Average TXOP = 8 ms



### Duty Cycling Example 3 - Average TXOP = 9.5 ms



### ETSI EN 301 893 Adaptivity Testing

Adaptivity testing focuses on the following three factors:

- Energy detect (ED)—Always tested by ETSI
- Adherence to maximum channel occupancy time (MCOT, also called TXOP limit)—Compliance by declaration is possible (does not have to be tested by ETSI)
- Medium access testing (also called random backoff)—Compliance by declaration is possible (does not have to be tested by ETSI)

#### ED Testing

ED testing verifies that transmissions are halted within the declared MCOT time, after an interfering signal is started when:

- the interfering signal is white noise, OFDM, or an LTE signal
- the interfering signal is received the tested device (UUT) at -72 dBm
- traffic loading is full buffer

The interfering signal must be detected with 100% reliability, so the practical ED setting may have to be less than -72 dBm

ED compliance is always tested, no compliance by declaration possible

#### MCOT Testing

MCOT testing verifies that transmissions without intermediate backoff do not exceed the announced maximum channel occupancy time (MCOT). When the device declares to make use of note 2 in table 8 in clause 4.2.7.3.2.4, the MCOT is 10 ms; otherwise the MCOT is 6 ms. Compliance by declaration is possible for MCOT referred to as option B, where option A is testing in the lab.

## Medium Access Testing

For medium access, the test verifies the distribution of the time between TxOPs (backoff) when there are no collisions. The backoffs must be distributed uniformly between 41 us and 176 us for TxOPs < 6 ms and between 41 us and 320 us for TxOPs > 6 ms (and < 10 ms).

$$41 \text{ us} = \text{AIFS} - 2 \text{ us} = 16 + 3*9 - 2$$

$$176 \text{ us} = \text{AIFS} + 15 \text{ slots} - 2 \text{ us} = 16 + 3*9 + 15*9 - 2 \text{ us}$$

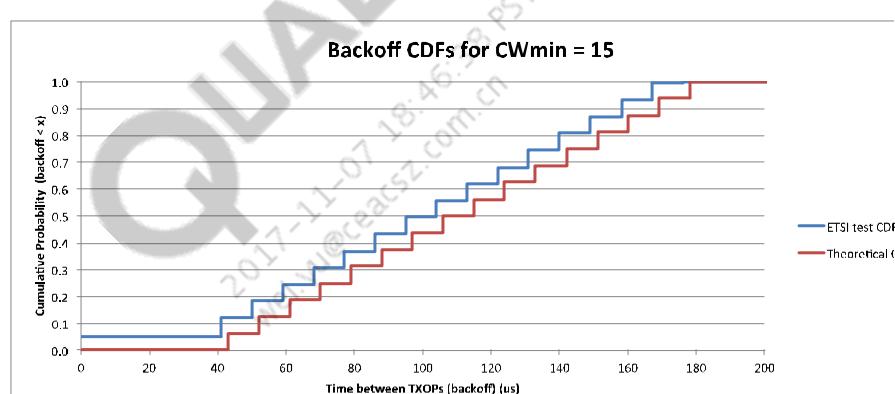
$$320 \text{ us} = \text{AIFS} + 31 \text{ slots} - 2 \text{ us} = 16 + 3*9 + 31*9 - 2 \text{ us}$$

2 us is subtracted to allow room for measurement and implementation errors

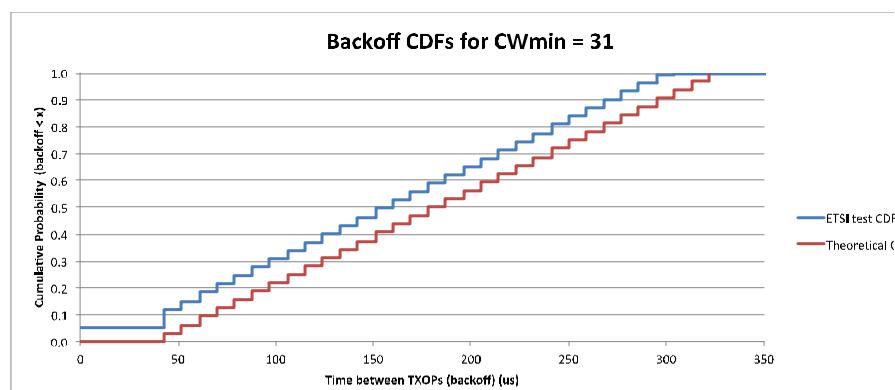
The distribution is tested by creating a CDF of the measured backoff gaps

Compliance by declaration is possible for medium access; referred to as option B, where option A is testing the lab

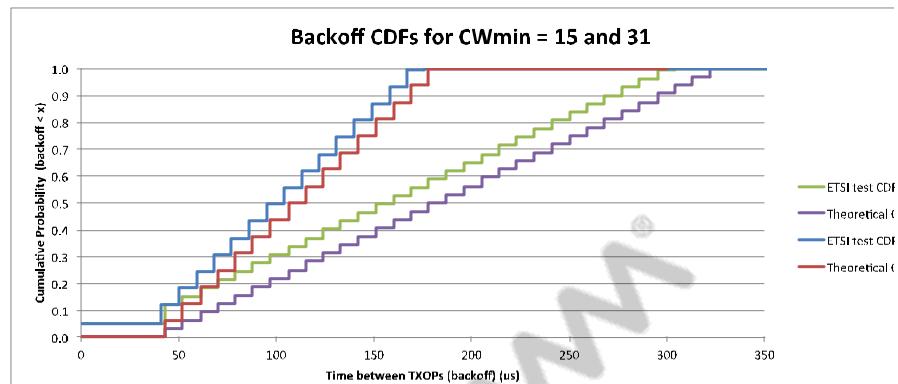
### Backoff CDFs for CW = 15 (TXOPs less than 6 ms)



### Backoff CDFs for CW = 31 (TXOPs between 6 ms and 10 ms)



### Backoff CDFs for CW = 15 and 31 (to see them on the same scale)



### ETSI Priority Classes

ETSI Priority Class mapping to EDCA Access Category (AC)

| ETSI Priority Class | 802.11 EDCA Access Category | 802.11 dot11QAPEDCATableIndex |
|---------------------|-----------------------------|-------------------------------|
| Priority Class 1    | AC_BK                       | 1                             |
| Priority Class 2    | AC_BE                       | 2                             |
| Priority Class 3    | AC_VI                       | 3                             |
| Priority Class 4    | AC_VO                       | 4                             |

## Client Access Parameters (ETSI)

Table 7: Priority Class dependent Channel Access parameters for Supervised Devices

| Class # | p | CW <sub>min</sub> | CW <sub>max</sub> | maximum Channel Occupancy Time (COT) |
|---------|---|-------------------|-------------------|--------------------------------------|
| 4       | 2 | 3                 | 7                 | 2 ms                                 |
| 3       | 2 | 7                 | 15                | 4 ms                                 |
| 2       | 3 | 15                | 1 023             | 6 ms<br>(see note 1)                 |
| 1       | 7 | 15                | 1 023             | 6 ms<br>(see note 1)                 |

NOTE 1: The maximum *Channel Occupancy Time* (COT) of 6 ms may be increased to 8 ms by inserting one or more pauses. The minimum duration of a pause shall be 100  $\mu$ s. The maximum duration (Channel Occupancy) before including any such pause shall be 6 ms. Pause duration is not included in the channel occupancy time.

NOTE 2: the values for p, CW<sub>min</sub>, CW<sub>max</sub> are minimum values. Greater values are allowed.

## AP Access Parameters (ETSI)

Table 8: Priority Class dependent Channel Access parameters for Supervising Devices

| Class # | p | CW <sub>min</sub> | CW <sub>max</sub> | maximum Channel Occupancy Time (COT) |
|---------|---|-------------------|-------------------|--------------------------------------|
| 4       | 1 | 3                 | 7                 | 2 ms                                 |
| 3       | 1 | 7                 | 15                | 4 ms                                 |
| 2       | 3 | 15                | 63                | 6 ms<br>(see note 1 and note 2)      |
| 1       | 7 | 15                | 1 023             | 6 ms<br>(see note 1)                 |

NOTE 1: The maximum *Channel Occupancy Time* (COT) of 6 ms may be increased to 8 ms by inserting one or more pauses. The minimum duration of a pause shall be 100 µs. The maximum duration (Channel Occupancy) before including any such pause shall be 6 ms. Pause duration is not included in the channel occupancy time.

NOTE 2: The maximum Channel Occupancy Time (COT) of 6 ms may be increased to 10 ms by extending CW to CW × 2 + 1 when selecting the random number q for any backoff(s) that precede the Channel Occupancy that may exceed 6 ms or which follow the Channel Occupancy that exceeded 6 ms. The choice between preceding or following a Channel Occupancy shall remain unchanged during the operation time of the device.

NOTE 3: The values for p, CW<sub>min</sub>, CW<sub>max</sub> are minimum values. Greater values are allowed.

## Client Access Parameters (WFA)

### 3.2.1 Default WMM parameters

Table 13 Default WMM Parameters

| AC    | CW <sub>min</sub>             | CW <sub>max</sub>             | AIFSN | TXOP Limit<br>(802.11b) | TXC<br>(802 |
|-------|-------------------------------|-------------------------------|-------|-------------------------|-------------|
| AC_BK | aCW <sub>min</sub>            | aCW <sub>max</sub>            | 7     | 0                       |             |
| AC_BE | aCW <sub>min</sub>            | aCW <sub>max</sub>            | 3     | 0                       |             |
| AC_VI | (aCW <sub>min</sub> +1)/2 - 1 | aCW <sub>min</sub>            | 2     | 6.016ms                 | 3.0         |
| AC_VO | (aCW <sub>min</sub> +1)/4 - 1 | (aCW <sub>min</sub> +1)/2 - 1 | 2     | 3.264ms                 | 1.5         |

Client parameters - numerical

| AC    | CWmin | CWmax | AIFSN | TXOP Limit<br>(802.11b) | TXOP Limit<br>(802.11a/g) |
|-------|-------|-------|-------|-------------------------|---------------------------|
| AC_BK | 15    | 1023  | 7     | 0                       | 0                         |
| AC_BE | 15    | 1023  | 3     | 0*                      | 0*                        |
| AC_VI | 7     | 15    | 2     | 6.016                   | 3.008                     |
| AC_VO | 3     | 7     | 2     | 3.264                   | 1.504                     |

## AP Access Parameters (WFA)

### A.4 WMM AP Default Parameter

It is recommended that the AP uses the default EDCA parameter listed in Table 15 and advertises the Table 13 values in WMM Parameter Elements.

**Table 15 Default EDCA Parameters for the AP**

| AC    | CW <sub>min</sub>              | CW <sub>max</sub>             | AIFSN | TXOP Limit (802.11b) | TXOP Lim (802.11a/g) |
|-------|--------------------------------|-------------------------------|-------|----------------------|----------------------|
| AC_BK | aCW <sub>min</sub>             | aCW <sub>max</sub>            | 7     | 0                    | 0                    |
| AC_BE | aCW <sub>min</sub>             | 4*(aCW <sub>min</sub> +1) - 1 | 3     | 0                    | 0                    |
| AC_VI | (aCW <sub>min</sub> + 1)/2 - 1 | aCW <sub>min</sub>            | 1     | 6.016ms              | 3.008ms              |
| AC_VO | (aCW <sub>min</sub> +1)/4 - 1  | (aCW <sub>min</sub> +1)/2 - 1 | 1     | 3.264ms              | 1.504ms              |

### AP parameters - numerical

| AC    | CWmin | CWmax | AIFSN | TXOP Limit (802.11b) | TXOP Limit (802.11a/g) |
|-------|-------|-------|-------|----------------------|------------------------|
| AC_BK | 15    | 1023  | 7     | 0                    | 0                      |
| AC_BE | 15    | 63    | 3     | 0*                   | 0*                     |
| AC_VI | 7     | 15    | 1     | 6.016                | 3.008                  |
| AC_VO | 3     | 7     | 1     | 3.264                | 1.504                  |

### Future Channel Access Requirements

The following flexibility might be required for channel access implementations in the future:

- CW at non-powers of 2
- CW that depends on the duration of a previous TxOP
- Non-random backoff equal to CW/2
- CW adaptation based on the number of times the backoff is interrupted (interruptions per transmission (IPT))

# 10 Memory and Performance

---

This chapter describes the memory and performance characteristics with WLAN AP operations, such as ART2 calibration, interrupt mitigation, SIFS Burst, Adaptive Noise Immunity (ANI), scatter/gather DMA, Airtime Fairness, Peer Flow Control Data Path, QCache, Firmware Code Sign, firmware authentication at host, WLAN LED Implementation, and preallocation of the required runtime memory.

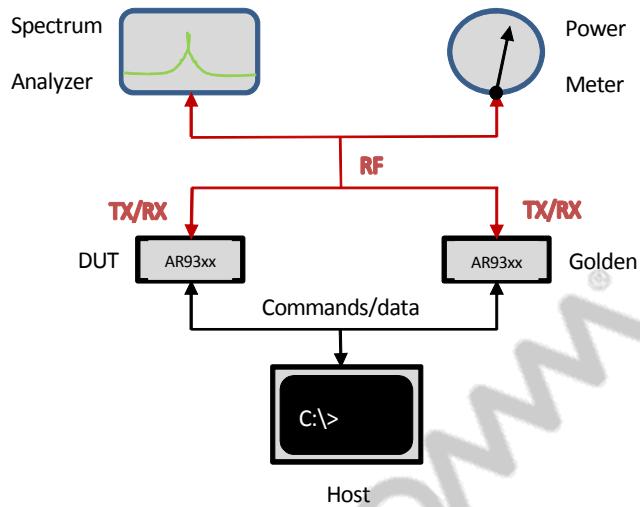
## 10.1 ART2 Calibration

Qualcomm Technologies reference designs require calibration in order to meet performance and regulatory requirements. This calibration process results in system-specific data that is stored in persistent memory and used at run time to compensate for system-to-system variations. These variations may be due to variations in components (including, but not limited to, product variations in board layout, and so on).

### 10.1.1 Theory of Operation

The calibration of QCA reference designs is accomplished by means of comparing the device-under-test (DUT) to a “golden” reference device.

The two devices are configured so that their RF transmitters/receivers are connected to each other via a network of RF plumbing that also connects to a spectrum analyzer and a power meter. See [Figure 10-1](#).



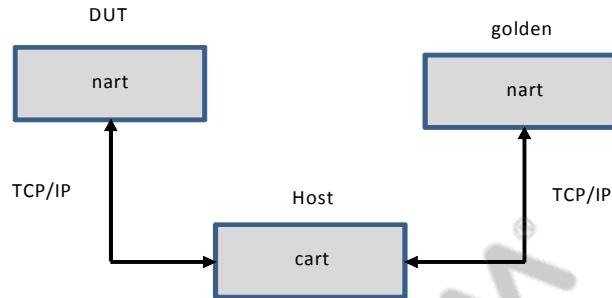
**Figure 10-1 Calibration Setup**

In this way, the transmitted RF power and frequency of the DUT can be measured, as well as any unwanted frequency spurs or power leakage into side-bands. In addition, by studying the statistics of packets sent from the golden device to the DUT, or vice versa, both Rx and Tx throughput and error rates can be collected.

#### 10.1.1.1 Implementation

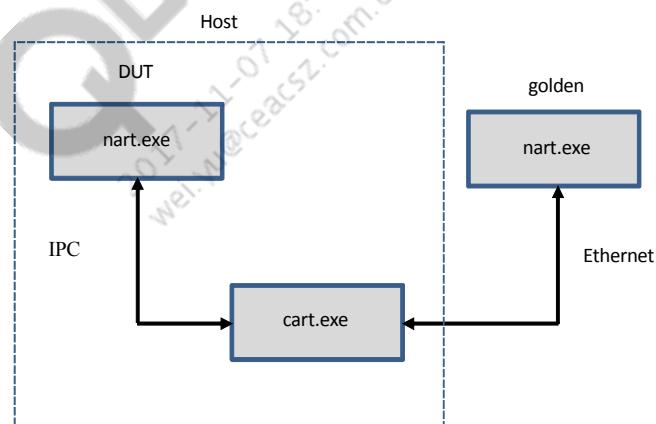
Calibration is executed by means of the ART2 software package. This package consists of three applications – **cart**, **nart** and the **artgui**, together with supporting libraries and drivers for both Windows and embedded Linux, as well as data that describes the various Qualcomm Technologies reference designs and the various calibration tests that need to be run for each reference design. All of this data is in the form of text files that can be configured by the customer to meet the needs of their particular design.

The calibration architecture is based on the **cart** and **nart** applications – **cart** corresponds to the host machine controlling the calibration process, while **nart** corresponds to the DUT and golden devices. This architecture is shown in [Figure 10-2](#).



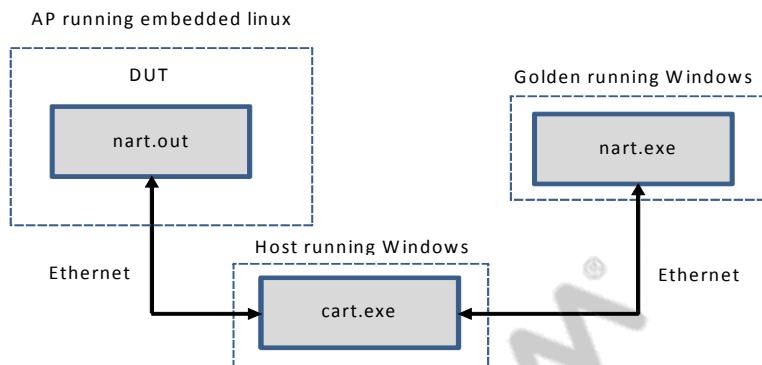
**Figure 10-2 ART2 Architecture**

If the DUT is a client card, then the **nart** process for the DUT runs on the same Windows box as the host application (**cart**). In this case the **nart** process for the golden device typically runs on a separate Windows box connected to the host via a dedicated LAN, as shown in [Figure 10-3](#).



**Figure 10-3 ART2 Client Setup**

If the DUT is an AP, then the **nart** process for the DUT typically runs as an application running on the embedded linux OS on the AP. In this case, **cart** runs as an application on the Windows host controlling the calibration process, and the **nart** process for the golden device is typically a Windows application running on the box controlling the golden device, as shown in [Figure 10-4](#).



**Figure 10-4 ART2 AP Setup**

Communication between the **cart** and **nart** applications is by means of TCP/IP sockets which write and read text strings. **Cart** instructs one of the **nart** applications to configure its device to receive or transmit by sending a string such as “rx <parameters>” or “tx <parameters>” respectively, where the parameters describe which radio to use and other radio parameters such as which chain to use, the frequency and data rate, the packet size and count, addresses to use, what parameters to measure, and so on.

In practice, the host-side **cart.exe** application is used to control the calibration process. **Cart.exe** is run in a DOS window, and executes scripts in the form of text files which describe the configuration of the system to be tested and the tests to be executed. These scripts are described in the next section.

### 10.1.1.2 Configuration

The calibration process begins with starting the **cart** application:

```
C:\myArt2>.\cart.exe -start mystart.art
```

Here, **mystart.art** is a script which describes the addresses of the equipment to be used in the calibration (power meter, spectrum analyzer, attenuators, and so on), as well as the pathlosses the RF signals experience traversing the RF plumbing between the various devices. When the start script is complete **cart** will print “Waiting for commands”.

At this point **cart** needs to be connected to the **nart** processes for the DUT and golden devices. This is accomplished with the **connect** command. This command (and all other commands) takes an **instance** argument that differentiates which device is intended. This command also takes a **host** argument that describes the address of the device intended. **Connect** is usually followed by a **load** command, which loads any existing calibration data from the device. These commands are as follows:

```
connect instance=0; host=dutaddr;
connect instance=1; host=goldenaddr;
load instance=0;
load instance=1;
```

In the above, `goldenaddr` is typically a LAN IP address, like 192.168.1.3. For a client card, `dutaddr` is typically `localhost`, while for an AP it is typically a LAN address like 192.168.1.2.

Once `cart` is connected to the `nart` processes, most calibrations can be executed by merely running the `test_flow` script, and providing the information for which `cart` prompts. Before the calibration tests themselves start running, `test_flow` will do three things: First, it will load a file that describes the system being tested. This file is typically named after the reference design, with a “.ref” suffix, for example, `AP113.ref`, and it contains various identification codes, regulatory domain information, descriptions of antenna chains and xPA and xLNA logic, and other things. Second, it will execute a script that describes the target powers for the board. This is typically named something like `tgt_pwr.art`. Third, it will execute a script that defines which tests will be run. This is typically called something like `test_flow_flags.art`. After this the tests themselves will execute, and write their results to both detailed log files and summary report files.

Examples of all the files and scripts described above can be found in the command folder of the ART2 release, and in sub folders named after various reference designs. For more information about ART2, refer to the *ART2 Reference Guide, Version 2.x (MKG-15527)*, and *ART2 Reference Guide, Version 4.x (MKG-17412)*.

QUALCOMM  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

## 10.2 Interrupt Mitigation/Moderation

### 10.2.1 Interrupts

The MAC interrupts the host CPU under several conditions. The most common interrupt causes are for received and transmitted packets. Less common causes for interrupts are timers (for beacons), error conditions in either transmit or receive (underrun/overruns), GPIO interrupts, stats counter overflows, and so on. Data transmit and receive account for the bulk of the interrupts generated by the MAC. Interrupts result in timely execution of critical code.

Consider the following case where the MAC is receiving a data stream at sustained rate of 300 Mbps. If the packet sizes were 1500 bytes each, this would translate to 25000 packets per second. If this were a TCP stream, this would result typically in an ACK being transmitted for every 2 packets that are received, which would imply that there are 12,500 packets transmitted by this MAC every second. Assuming that each packet received or sent would generate an interrupt, this would be 37500 interrupts per second, or one interrupt every 27 microseconds.

In a typical Wireless MAC, beacons are transmitted (or received) every 100 ms. These packets account for only an additional 10 interrupts a second which is insignificant compared to the transmit and receive data interrupts. Similarly, transmitting and receiving management packets cause a relatively insignificant number of interrupts.

Each interrupt causes a break in the thread of execution on the CPU, which in turn causes churn in cache contents. The combined effect of interrupts at the rate of one every 27  $\mu$ s is a lot of CPU execution time being wasted in waiting for various memory banks to refresh, contexts to change, and so on, and results in less real work being done.

To avoid this loss of CPU cycles in wasted work, a scheme is implemented where the number of interrupts that are generated by the device is reduced. However, on each interrupt more packet processing is done.

### 10.2.2 Mitigation: A Simple Implementation

In its simplest form, interrupt mitigation is a simple periodic poll function. When a timer fires, the relevant code (receive or transmit handler) processes packets that have been received since the last time the timer went off. The timer can be fired by a software method, such as an OS specific timer. However, OS timers are not guaranteed to be accurate—a timer that is requested at a 1 ms interval could really only fire at a 2 ms interval. To avoid elasticity in OS timers, this can be implemented by having the MAC hardware generate interrupts periodically (assuming that the MAC has a timer that is capable of interrupting the host). Device interrupt handler routines run at high priority and are guaranteed to execute with minimum latency. Receive and transmit completion functions are executed every time the MAC timer interrupts the CPU.

When such a timer interrupt is received, there are several packets that have been either received or transmitted. All accumulated packets are now processed in the handling function (DPC/Tasklet). The context switch that happens to process the timer is now amortized over several packets instead of each packet generating an interrupt and incurring the overhead. This in general results in lower CPU utilization.

### 10.2.3 Mitigation: Unintended Consequence

Another effect of interrupt mitigation is the latency incurred on every packet completion. Since packets now have to wait for a considerably longer time before they are detected and passed up to upper layer protocols, system response goes down. If packets are being transmitted or received at large intervals, each completion is only processed upon detection by the delayed interrupt.

To reduce latency, the MAC hardware implements a mechanism whereby the latency can be limited, while at the same time allowing for better CPU utilization when there is a steady stream of traffic. This is described in the next section.

### 10.2.4 MAC Mitigation Support

#### 10.2.4.1 Receive Completion Interrupt Mitigation

The MAC hardware provides special functionality to reduce interrupts on the receive side. Interrupts are normally issued by hardware upon reception of a packet after it has been moved into host memory. There are two timers implemented by the MAC: First timer threshold and last timer threshold. Both these timers are started upon reception of the first packet. If a second packet is received before the last timer expires, the last timer is reset and restarted. The restarting of the last timer continues until the first timer expires, when an interrupt is forced to be issued to the CPU.

The last timer limits latency when traffic is slow, and packets are being received at intervals greater than the value of last timer. The first timer limits the number of packets that can be accumulated before an interrupt is raised to the CPU and reduces latency.

#### 10.2.4.2 Transmit Completion Interrupt Mitigation

Interrupt mitigation on the transmit side in 802.11n is already built into the link layer protocol when aggregate transmission is used. Aggregate transmission involves sending multiple packets (subframes in 802.11n) bound for the same destination with no 802.11 ACK for each individual subframe. These subframes are separated by delimiting symbols and there is very little overhead in the delimiters. The number of delimiters is negotiated and can be as small as a single delimiter between subframes. As many as 64 subframes can be part of an aggregate. Typically, Qualcomm Technologies drivers limit the aggregate size to 32 subframes. The hardware is designed to generate an interrupt on the reception of a Block ACK when it is received. This effectively brings down the transmit completion interrupt rate to 1 per 32 packets and in most cases, nothing more is needed to mitigate Transmit Completion Interrupts.

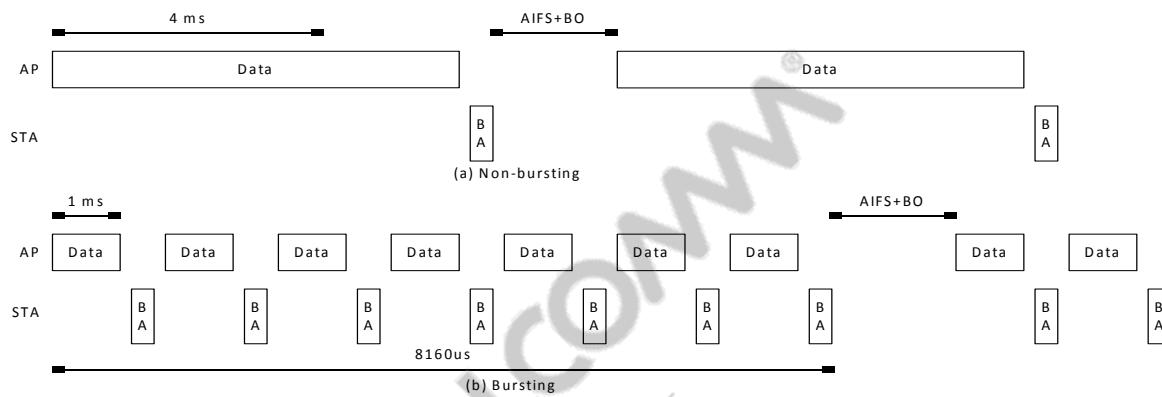
In addition to this, hardware provides functionality to reduce interrupts further. This mechanism is identical to what is described in the preceding section.

## 10.3 SIFS Burst

SIFS spaced bursting is done by sending AMPDUs in SIFS period following the Block ack from a station. This ensures that medium is occupied by transmission more and increases the MAC efficiency.

### 10.3.1 Theory of Operation

This kind of back-to-back transmits within SIFS period can be done for a controlled transmit opportunity (TxOP) period, which is 12 successive AMPDUs (maximum 12 ms) in current implementation. The result of this will be higher throughput over non-SIFS burst in OTA environment. This is pictorially illustrated in [Figure 10-5](#).



**Figure 10-5 SIFS Burst Overview**

### 10.3.2 Implementation

The SIFS bursting is implemented in firmware using the VMF bursting bit in the TX PPDU descriptor. The HW MAC transmits the next frame in the HW queue using SIFS bursting if the VMF bit in the descriptor is set. The HW MAC ensures that all the frames in a burst transmission belong to same AC. If there is no frame in HW queue corresponding to an access category then burst will be broken and MAC will re-contend (AIFS+random backoff) for medium access.

Currently When SIFS Burst is enabled, AMPDU are limited to a maximum of 1024us. Also SIFS burst will be done for a maximum consecutive 12 AMPDUs. Once this limit is reached, or if no frames available from same AC, then SIFS burst will be broken by unsetting VMF bit. Also 2 msec until Beacon TBTT, the firmware will not set VMF bit to allow beacons to get transmitted on time and avoid bursting to delay the beacon.

Currently SIFS Burst is enabled by default in some solutions. All other solutions have this disabled by default.

The SIFS bursting can be dynamically enabled or disabled using burst/get\_burst command.

## 10.4 Adaptive Noise Immunity (ANI)

Adaptive Noise Immunity is the algorithm used by the WLAN software to make the baseband receiver immune to particular types of noise by dynamically tuning the receive parameters.

### 10.4.1 Theory of Operation

ANI is used to reduce the false packet detect rate when the WLAN hardware has to operate in the presence of a source of interference such as a spur or another non-WLAN transmitter in the same channel/band.

The receiver in the WLAN hardware detects a frame by searching for a repeating sequence (the STFs of the 802.11 frame preamble). It does this by correlating the time of the received signal with a delayed version of the signal. This is called weak signal detection. When a source of interference such as a spur or a non-WLAN transmitter is present in the same channel on which the WLAN receiver is receiving frames, the WLAN receiver may detect it as a frame if the interference has periodicity in its signal. Subsequent receiver checks (such as the LTFs or SIGNAL field) will detect that the interference signal is not a valid WLAN frame and the receiver will restart its receive state machine. This is called a false detection which is recorded as an OFDM or CCK PHY error (hardware can be configured to count PHY errors). Each false detection takes at least 8  $\mu$ s (duration of the STFs) and may take longer, as it takes time to detect an invalid frame. The false detection impacts the WLAN performance in two ways:

- Because the receiver has spent time trying to detect a frame that was actually interference, it is unable to transmit at the same time—so there is a loss of air time which could have been spent transmitting a frame.
- The false detection could prevent the reception of a legitimate WLAN frame being transmitted at the same time, or slightly later (during the 8  $\mu$ s when the WLAN receiver is false detecting).

Therefore, if the false detects are very frequent (which will be the case if it is a spur or a continuous wave interferer), performance will be impacted. As indicated above, both transmission and reception are affected.

ANI makes the receiver immune to such sources of interference, by reducing the receiver sensitivity. If the interference signal is sufficiently low power, then it will no longer trigger false detections. If the interference is not detected, the WLAN receiver can receive valid incoming frames or transmit frames.

The drawback of ANI is that it impacts receiver sensitivity; that is, the range. Note that ANI can only help if the interferer signal power is low enough to be rejected.

### 10.4.2 Implementation

When ANI is enabled, the MAC counters are set such that they count the number of OFDM and CCK PHY errors. At periodic intervals of time, the ANI state machine checks these registers. If the error count in an interval of time exceeds a threshold, the ANI state machine changes its state to a higher immunity level. When the ANI state machine changes its state, it sets the receiver signal detection thresholds in the baseband registers. There are seven immunity levels: the highest immunity level has the greatest immunity to interference and the least sensitivity. If the PHY error count in an interval of time drops below a threshold, the ANI state machine changes to a lower immunity level.

### 10.4.2.1 ANI code

The code is in the HAL directory

- wlan/drivers/hal/ar9300/ar9300\_ani.c
- wlan/drivers/hal/ar9300/ar9300.h
- wlan/drivers/hal/ah.h
- wlan/drivers/hal/ah\_internal.h

The following sections describe ANI operation for AR9300 and later chips. However, the ANI implementation is mostly identical as for AR5416 and earlier 802.11n chips)

### 10.4.2.2 ANI Data

The data structures are defined in ar9300.h

**struct ar9300\_ani\_state:** This data structure contains the ANI state variables, the ANI configuration parameters and the ANI statistics

#### ANI state

- Ofdm\_noise\_immunity\_level: This is the current immunity level for OFDM false detection
- cck\_noise\_immunity\_level: This is the current immunity level for CCK false detection
- spur\_immunity\_level: This is the current immunity level for spurs
- firstep\_level: This is current threshold for in-band power rise required to validate weak detection

#### Configuration parameters

- ofdm\_weak\_sig\_detection\_off: This variable gets set during ANI operation when the RSSI of the received PHY errors is above a particular threshold. It turns off weak signal detection, which prevents a weak signal from getting engaged unnecessarily because of an interferer.
- mrc\_cck\_off: This variable gets set if CCK noise is detected. It turns off MRC CCK detection.
- ofdm\_trig\_high, ofdm\_trig\_low: These variables control the thresholds at which the ANI state machine changes state to a higher or lower level of immunity. The OFDM PHY error rate is compared to these variables to decide whether to raise/lower the immunity level.
- cck\_trig\_high, cck\_trig\_low: These variables control the thresholds at which the ANI state machine changes state to a higher or lower level of ODFM noise immunity. The CCK PHY error rate is compared to these variables to decide whether to raise/lower the CCK noise immunity level.
- rssi\_thr\_low, rssi\_thr\_high: These variables also control when the ANI state machine changes state to a higher or lower level of immunity. In addition to the two variables above, the RSSI of the PHY error is compared to these thresholds to decide whether to raise/lower the immunity level.
- Ini\_def: This structure contains the default values for the ANI signal detection registers. The signal detection registers include

- m1\_thresh\_low, m1\_thresh, m1\_thresh\_low\_ext, m1\_thresh\_ext
- m2\_thresh\_low, m2\_thresh, m2\_thresh\_low\_ext, m2\_thresh\_ext, m2\_count\_thr, m2\_count\_thr\_low
- firststep, firststep\_low
- cycpwr\_thr1, cycpwr\_thr1\_ext

The m1thresh registers control the threshold for self correlation detector in the first stage of the correlation in weak signal detection.

The m2thresh registers control the threshold for the self correlation detector in the second stage of the correlation in weak signal detection.

The firststep registers control the threshold beyond which the step in-band power rise must exceed for weak signal detection to engage.

The cycpwr\_thr registers control a threshold for cyclic power. If cyclic power is below this threshold, the signal is not detected during weak signal detection

- Statistics:
  - tx\_frame\_count, rx\_frame\_count, cycle\_count: These are counts of the hardware TxFrame, RxFrame and cycles obtained from MAC registers.
  - ofdm\_phy\_error\_count, cck\_phy\_error\_count: These are variables which hold the OFDM and CCK PHY error counts for the last collection period.
  - ah\_ani\_period: This variable controls two events:
- When the errors counters are checked to see if the immunity level needs to be raised (one period)
- When the error counters are checked to see if the immunity level needs to be lowered (five periods)

The default value is 1000 ms.

### 10.4.2.3 ANI Functions

The functions are within the HAL:

#### Attach functions

- ar9300\_ani\_attach
- ar9300\_ani\_detach

The attach function initializes the ANI state variables to the default values. In addition, it configures the hardware MAC error registers to count CCK and OFDM PHY errors. The detach function turns off collection of PHY errors by hardware.

#### Initialization functions

- ar9300\_ani\_initdefaults()

The function ar9300\_ani\_initdefaults() reads the signal detection register values and initializes the internal ANI state variable ini\_def, which stores the default signal detection register values.

- ar9300\_ani\_restart()

The ani\_restart() function initializes the ANI PHY error counters and restarts the counting. It does not modify the current ANI immunity level.

- ar9300\_ani\_reset()

The ani\_reset() function sets the immunity level to the default value. It calls ani\_restart to initialize the ANI PHY error counters.

### **Immunity Level Control functions**

- ar9300\_ani\_set\_ofdm\_noise\_immunity\_level()

This function takes as argument the new desired OFDM noise immunity level, checks the current spur immunity level and the firststep noise level, and sees if they correspond to the expected values. If they do not, it calls the ani\_control() function to set the spur immunity level and firststep immunity level to the expected values. In addition it checks if the OFDM weak signal detection needs to be turned on or off.

- ar9300\_ani\_set\_cck\_noise\_immunity\_level()

This function takes as argument the new desired CCK noise immunity level, and checks the firststep noise level to see if it corresponds to the expected value. If it does not, it calls the ani\_control() function to set the firststep immunity level to the expected values. In addition it checks if the MRC CCK detection needs to be turned on or off.

- ar9300\_ani\_lower\_immunity()

This function lowers the OFDM and CCK immunity level by 1 level.

- ar9300\_ani\_control()

This function controls the ANI immunity settings. It takes as argument a command and a parameter. The command and its effects are as follows:

- Weak Signal Detection: Turn On/Off, impacts false detection rate
- FIR Step Level: Increase/Decrease the level, impacts false detection rate
- Spur Immunity Level: Increase/Decrease immunity to spurs by adjusting the cyclic power threshold
- MRC CCK: Turn On/Off, impacts false detection rate

Depending on the command and the associated parameters, ANI registers are set to the appropriate values.

### **ANI trigger functions**

- ar9300\_ani\_ofdm\_err\_trigger()

This function calls ar9300\_ani\_set\_ofdm\_noise\_immunity\_level() if the OFDM noise immunity level is below the maximum level. It also checks that ANI is enabled.

- `ar9300_ani_cck_err_trigger()`

This function calls `ar9300_ani_set_cck_noise_immunity_level()` if the OFDM noise immunity level is below the maximum level. It also checks that ANI is enabled.

- `Ar9300_ani_process_mib_intr()`

This function is used if the PHY error frames are used as a trigger for ANI. This method is not used by default. Instead the PHY error counters are used.

- `Ar9300_ani_ar_poll()`

This function reads the PHY error count registers and finds the listen time and calculates the PHY error rate. If the listen time is greater than a threshold and the PHY error threshold (CCK or OFDM) is greater than a threshold, then it calls the corresponding OFDM/CCK error trigger functions (above) with the new level as the argument. If the listen time is greater than another threshold and the PHY error threshold (CCK or OFDM) is lower than a threshold, then it calls the corresponding immunity lower function.

- `Ar9300_ani_get_listen_time()`

This function returns the effect time spent listening for PHY errors since the last PHY error counter check. It subtracts the time spent transmitting or receiving frames from the total cycle count.

#### 10.4.2.4 Control flow

ANI operation is triggered through the `ar9300_ani_ar_poll()` function, which gets called periodically through the device calibration function. This function in turn calls the needed functions to increase/decrease the immunity level.

#### 10.4.2.5 API

The API functions are:

- `ah_rx_monitor()`

This function takes an `ath_hal` pointer, a node stats pointer, and a channel pointer as arguments. Its maps to the `ar9300_ani_ar_poll()` function. It is called from the LMAC periodically by the `ath_calibrate` function.

- `ah_process_mib_event()`

This function is called from the LMAC when it receives a PHY error interrupt. It maps to the `ar9300_process_mib_intr()` function.

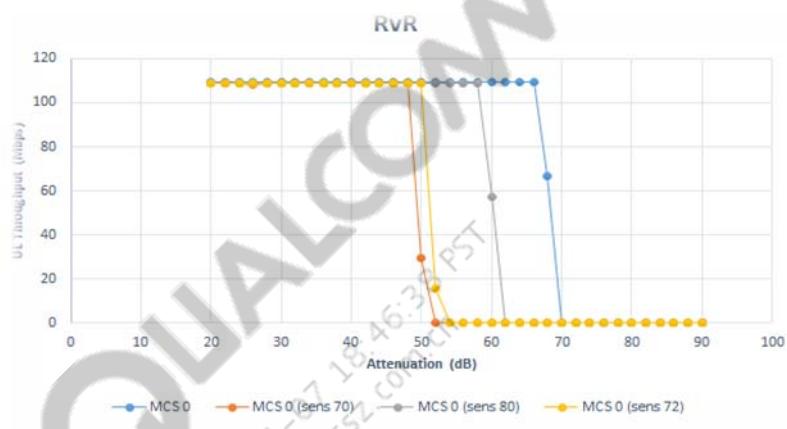
#### 10.4.2.6 Configuration

There are no CLI commands to change the default ANI operations.

## 10.5 Disable ANI to control packet detection threshold

In high-density WLAN deployments, an AP locks to receive packets from faraway cochannel BSS. In such environments, a requirement exists to set the packet detection threshold to any value between -100 dBm and -60 dBm in increments of 2 dB. ANI periodically (1 second) checks the PHY error count and adjusts the receive sensitivity dynamically. When the receive sensitivity is set to a particular threshold, ANI must be disabled to meet the sensitivity requirement. Depending on the received PHY errors count, ANI algorithm changes the Rx sensitivity dynamically. Therefore, ANI is disabled after setting the required sensitivity level to meet the requirement.

The following figure illustrates the rate vs range (RvR) test results in the uplink direction with different sensitivity levels in cabled environment:



An option is not present to disable this feature. As a workaround, setting the default sensitivity level (-95 dBm) causes ANI to be disabled. The configurable minimum receive sensitivity is between -95 dBm and -100 dBm.

## 10.6 Airtime fairness

The Airtime Fairness (ATF) requirement primarily focuses on scheduling fairness for transmission of traffic from Access Point (AP), and efficient Wi-Fi bandwidth utilization. Its algorithm does not deal with transmission of frames from other clients.

### 10.6.1 ATF requirement

- **SSID-based airtime allocation** – The airtime management configuration on a per-SSID basis shall be configured as a percentage of available airtime, to be allocated to an SSID or group of SSIDs.
- **Equal airtime allocation** – The clients within the BSS should get equal air time allocation.
- **Unused airtime redistribution** – Redistribute unused airtime. The ATF algorithm should be able to share the unused bandwidth from home users to the hotspot user, if there is no traffic in the home network.

- **Per station airtime allocation** – This mechanism shall primarily be used to make sure that the STAs are allocated enough bandwidth to perform their respective tasks (such as video streaming).

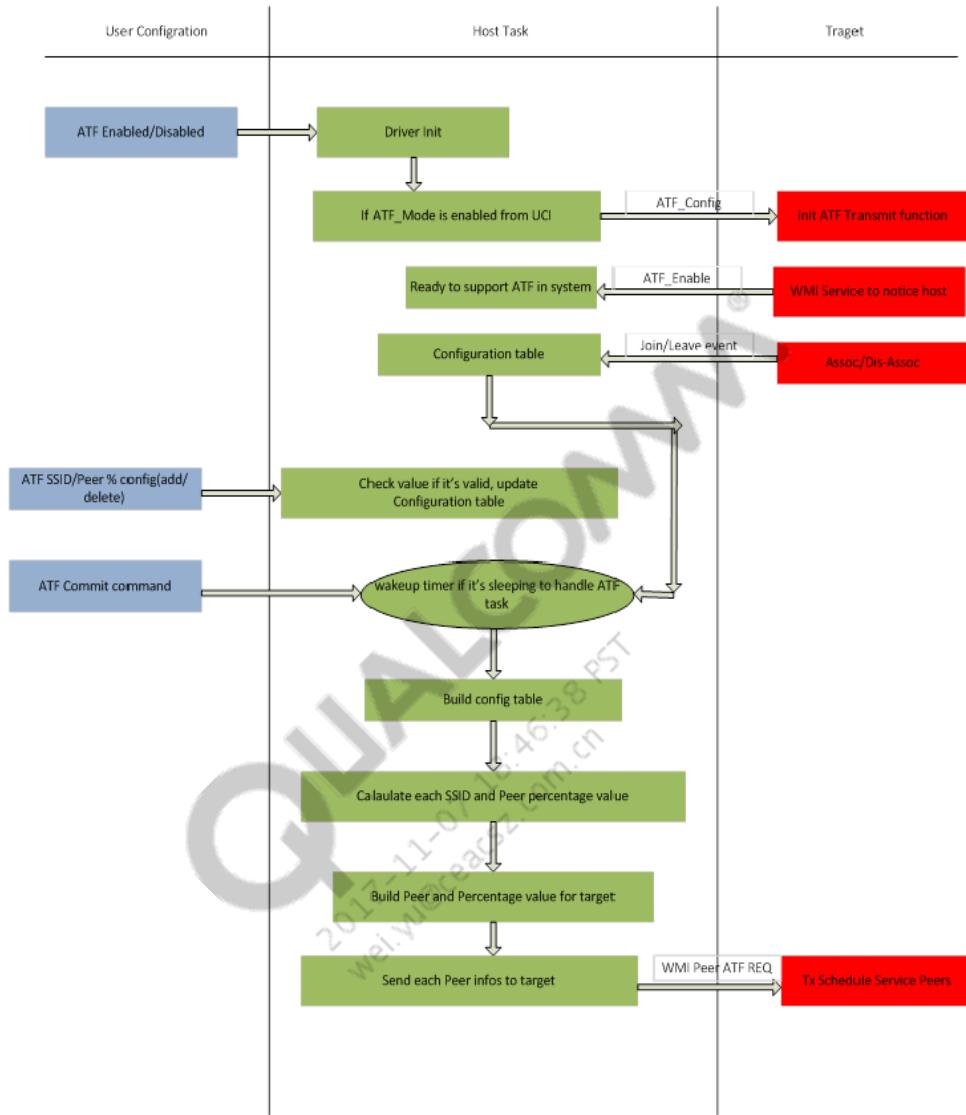
## 10.6.2 ATF design for offload architecture

### 10.6.2.1 Host and firmware design

Host majority task focus on getting user configuration, allocating table for all SSID, peer and percentage value, calculating each peer entry of ATF, and configuring resources for each peer that need to be send firmware. Host needs to control the interaction between host and firmware. Due to the event of set-up allocation table is triggered by different way either user input or peer join/leave, host generates a timer to do this flow for each event until host passes all configuration resource to firmware via WMI command.

#### Sequence Flow

Figure 10-6 shows the flow sequence between host and target plus user interface.

**Figure 10-6 Sequence Flow**

## Implementation

The host driver implements a few portions to achieve this feature. The user input configuration either adds/deletes SSID or peer MAC address into the configuration table; the peer's join/leave also needs to be updated in the configuration table. The ATF task timer involves majority tasks. The WMI command handles the interaction between host and firmware.

- User inputs SSID and percentage value via the IOCTL interface. They are filled into the global configuration table and the driver needs to make mark label for it in the field `atfcfg_set.vap[x].cfg_flag` and put the configuration value into the global data structure.
- User inputs peer MAC address and percentage value via IOCTL interface. They are filled into the global configuration table, and drivers needs to make mark label for it in field `atfcfg_`

set.peer\_id[x].cfg\_flag, atfcfg\_set.peer\_id[x].index\_vap[x] and put the configuration value into global data structure.

- User removes SSID or peer MAC address from the configuration table via IOCTL; then the driver resets the fields from the table for cfg\_flag, cfg\_value, vap\_index, and so on.
- Peer's association or disassociation send join/leave event to the driver. It adds or removes peer MAC addresses from the global table and wakes up the ATF timer task.
- User issues the “iwpriv athx commitatf 1/0” command to effect the input setting; this wakes up the ATF time schedule task.
- Timer task is split into three sub-tasks, build\_atf\_alloc\_tbl(), cal\_atf\_alloc\_tbl(), and build\_atf\_for\_fm().

See [Figure 10-7](#) and [Figure 10-8](#).

- Build\_atf\_alloc\_tbl() is responsible for building up the relation between peer and SSID, collecting all peer information and looking up the ATF config VAP table. It checks if peer's SSID is in this table; if it exists, the VAP index is placed in the peer's vap\_index field. Else, 0xff is placed in the vap\_index field.
- Cal\_atf\_alloc\_tbl() is based on the relations set up between peers and VAPs. It calculates every VAP and associated peer's ATF percentage number; the table keeps each peer's calculated number.
- Build\_atf\_for\_fm() derives every peer's MAC address and the corresponding calculated ATF's number from the table, and builds and passes configuration information to the firmware via WMI interface.
- Two user command interfaces could be used to provide all SSIDs (VAPs) and peers ATF percentage numbers.

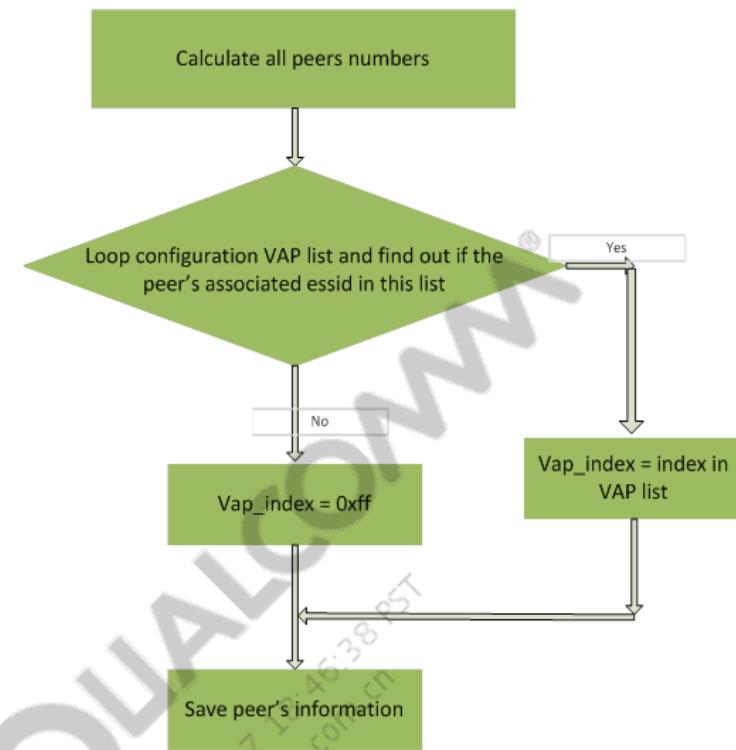
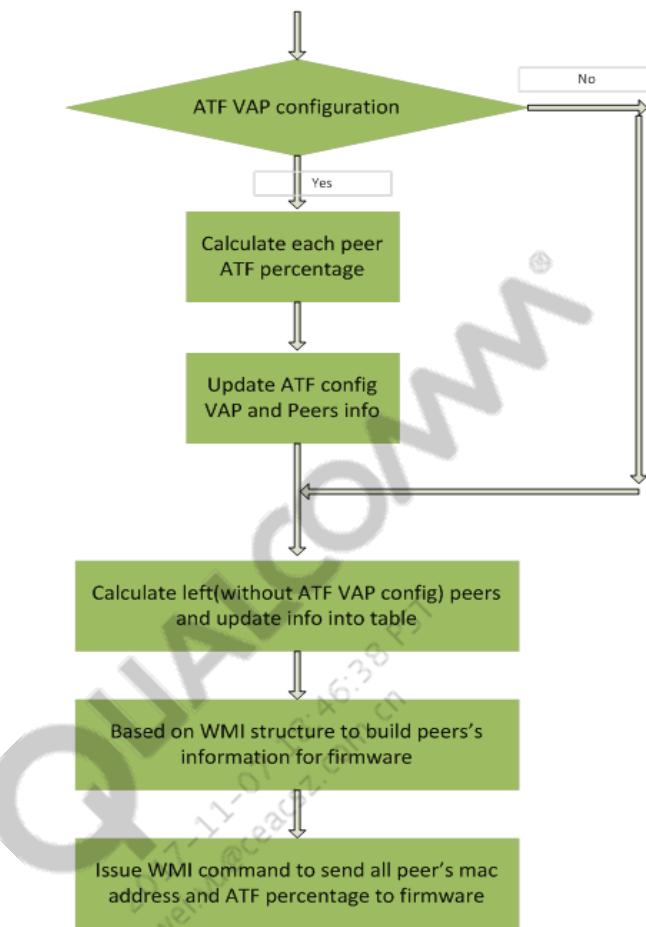


Figure 10-7 ATF Implementation Part 1



**Figure 10-8 ATF Implementation Part 2**

#### 10.6.2.2 Host/Target Interaction

The host handles all the conversion from the user percentage to per mille (1 part in 1000) distribution across all the clients. Every time the distribution changes, the host syncs the per mille table for all stations with the target. The per mille between host and target enables handling fractional percentage without dealing with floating point arithmetic.

#### 10.6.2.3 ATF CLI commands

For details on the ATF CLI commands, see *AP 10.4 Command Line Interface (CLI) User Guide* (80-Y8052-1).

## 10.6.3 ATF design for direct attach architecture

### 10.6.3.1 Strict queue and Fair queue ATF algorithm

Airtime Fairness implements 2 scheduling algorithms which are mutually exclusive: Strict queue and fair queue algorithm. The Strict queue algorithm follows strict airtime allocation as configured by the user and does not try and utilize any unused bandwidth. The fair queue algorithm on the other hand guarantees the configured airtime in congested environments and it also utilizes any unused bandwidth. The difference can be best explained by the following example.

Consider 2 clients, Client 1 and Client 2 for which the following airtime is configured

**Table 10-1 Strict queue and fair queue ATF algorithm**

| Client name | Airtime configured |
|-------------|--------------------|
| Client 1    | 60                 |
| Client 2    | 40                 |

- In a congested environment, i.e., when data is sent to both clients at the peak rate, Client 1 is guaranteed 60% of the airtime and Client 2 is guaranteed 40% of airtime
- In a case where Client 2 is idle and data is sent at peak rate to Client 1: with Strict queue algorithm, client 1 is allotted with only 60% of airtime and the remaining 40% is reserved for Client2. With Fair queue algorithm, Client 1 will try and utilize the unused 40% allotted to Client 2. So in-effect Client 1 will be allotted almost 100% of the airtime as long as Client 2 is idle.

Airtime fairness implementation for the direct attach architecture can be broadly classified as follows:

- ATF table update mechanism
- ATF scheduler

The ATF table update mechanism implementation differs for strict queue and fair queue algorithms. However, the ATF scheduler implementation is the same for both strict and fair queues.

### 10.6.3.2 ATF table update method for strict queue scheduling algorithm

The Wi-Fi driver maintains the ATF Table based on the user configuration. The user configures the airtime to be allotted for a VAP or a specific peer. The ATF table update logic takes the user input and converts the percentage provided to tokens. The scheduling is based on the token availability. Each ‘Token’ is a unit of time and each token equates to 1 micro second.

The ATF table update logic is inherited from the offload implementation. Refer to [Section 10.6.2](#) details.

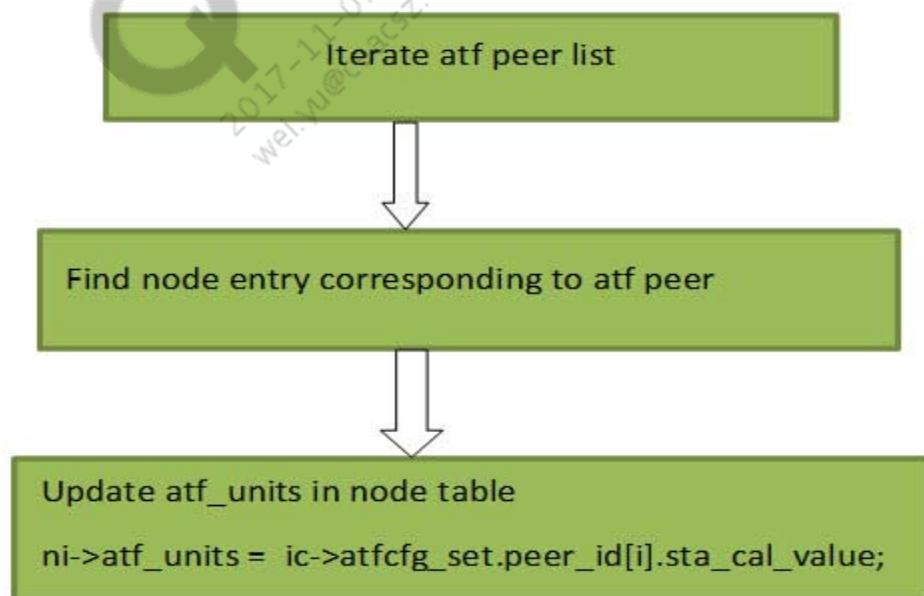
The following are the direct-attach specific changes:

- After the ATF table is updated following a user configuration or client join/leave, update\_atf\_nodetable routine iterates through each peer entries in the ATF table and finds the corresponding entry in the node table (struct ieee80211\_node). The routine then reads the ATF unit allotted to the peer (ic->atfcfg\_set.peer\_id[i].sta\_cal\_value) and updates this in the node structure.
- A timer, atf\_tokenalloc\_timer configured to run every 200 milliseconds, iterates through the node table. This timer routine converts ATF units to tokens that can be passed on to the scheduler. For each valid node entry in the node table, the timer reads the ATF unit and calls the routine ieee80211\_distribute\_atf\_txtokens to distribute tokens. The tokens are passed on to the LMAC layer. Tokens are calculated using the simple formula :
  - $Tx\ tokens = ATF\_unit * (Timer\ Interval\ in\ micro\ seconds) / ATF\ DENOMINATION$
  - where ATF DENOMINATION is set to 1000. The choice to use 1000 (1 part of 1000) rather than percentage (1 part of 100) will help us to avoid dealing with floating point arithmetic and thus less approximation.

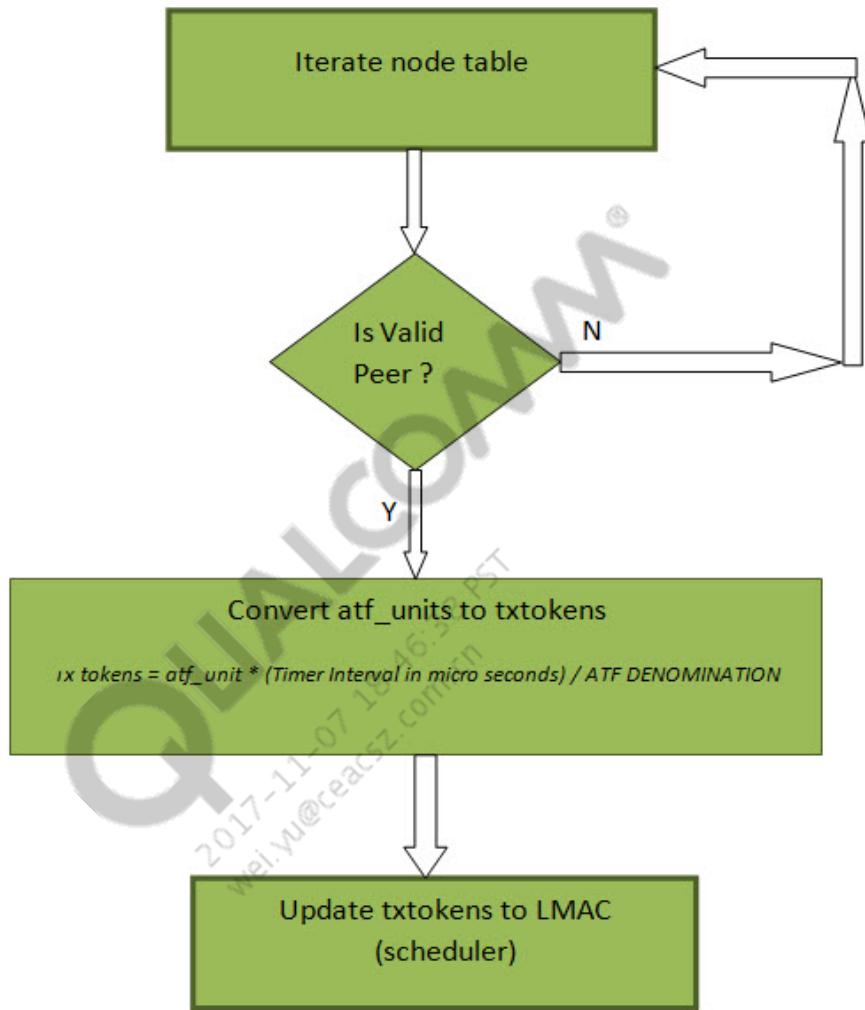
Thus, for a peer with 100% ATF allotted, txtokens would be 200000 ( $1000 * (200 * 1000) / 1000 = 200000$  usec)

The table update mechanism is part of Wi-Fi UMAC layer.

#### **update\_atf\_nodetable routine**



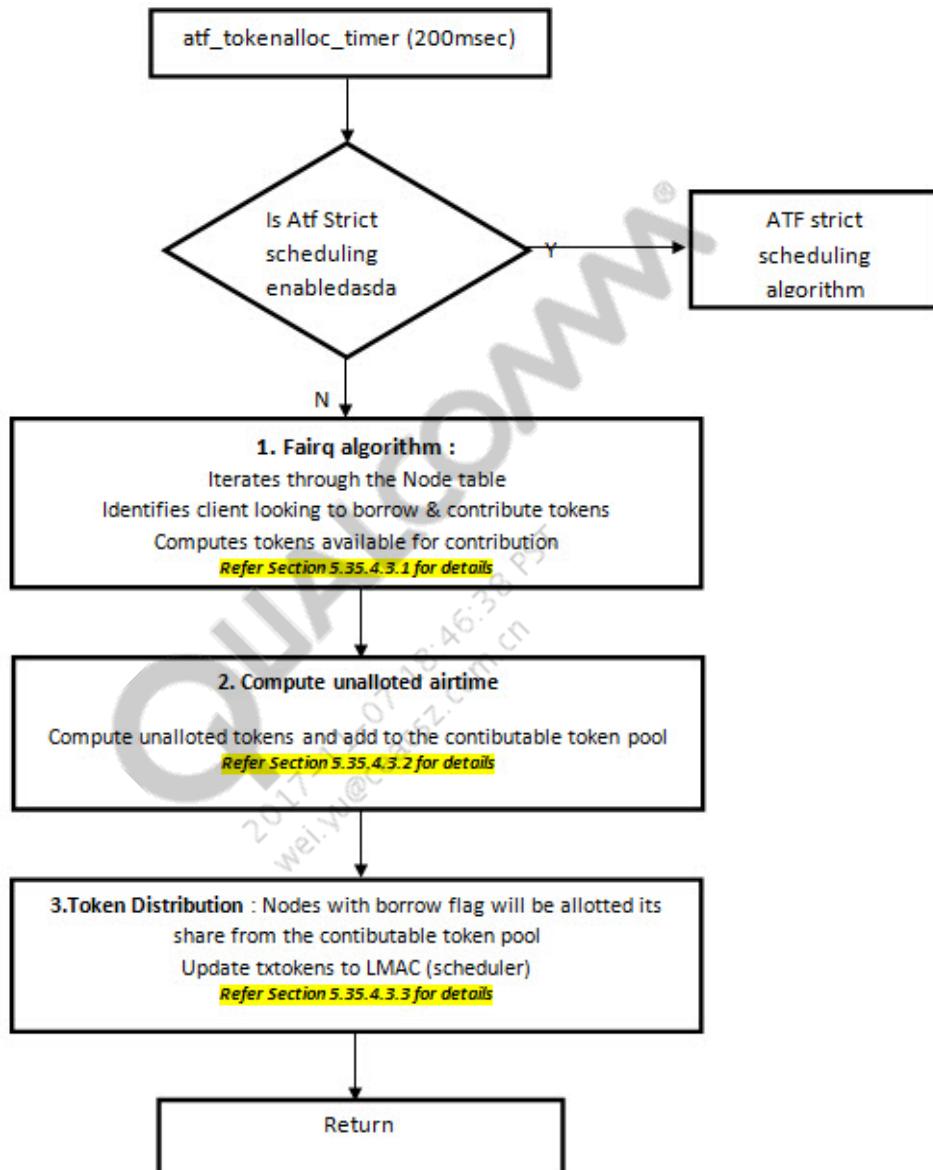
### Token allocation timer



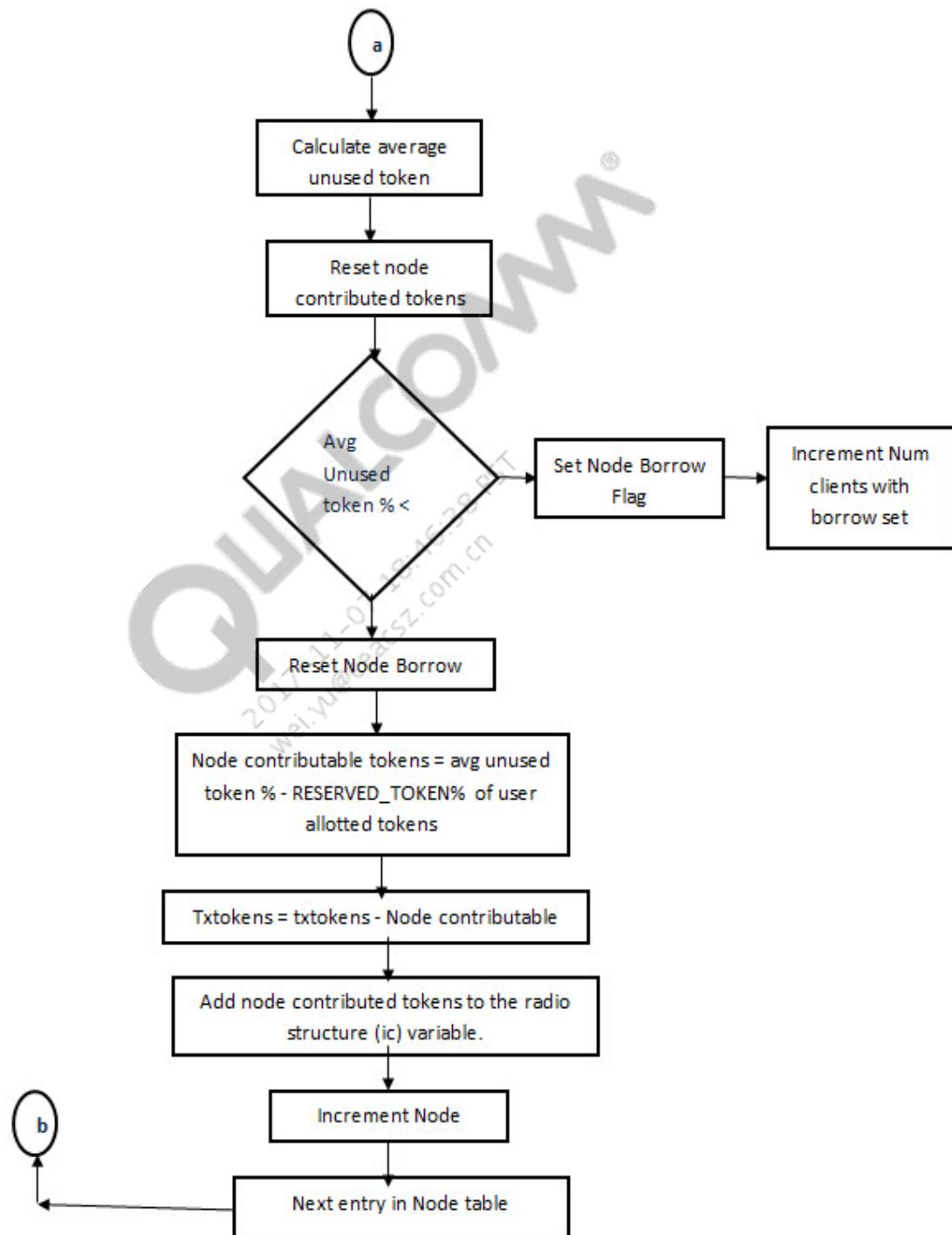
#### 10.6.3.3 ATF table update method for fair-queue scheduling algorithm

The fair queue algorithm maintains a history/traffic pattern to identify the bandwidth usage of each nodes. This history/pattern is consulted to mark a node as ‘borrow’ enabled or ‘contribute’ enabled. If the node pattern shows up that the average unused tokens is less than a set threshold, the node is marked to ‘borrow’ tokens from other nodes. Similarly if the node pattern shows up that the average unused node tokens are more than a set threshold, the unused tokens are added in the contribute-enabled token pool, which can then be distributed to nodes with borrow-enabled. The algorithm is illustrated in the following flowchart:

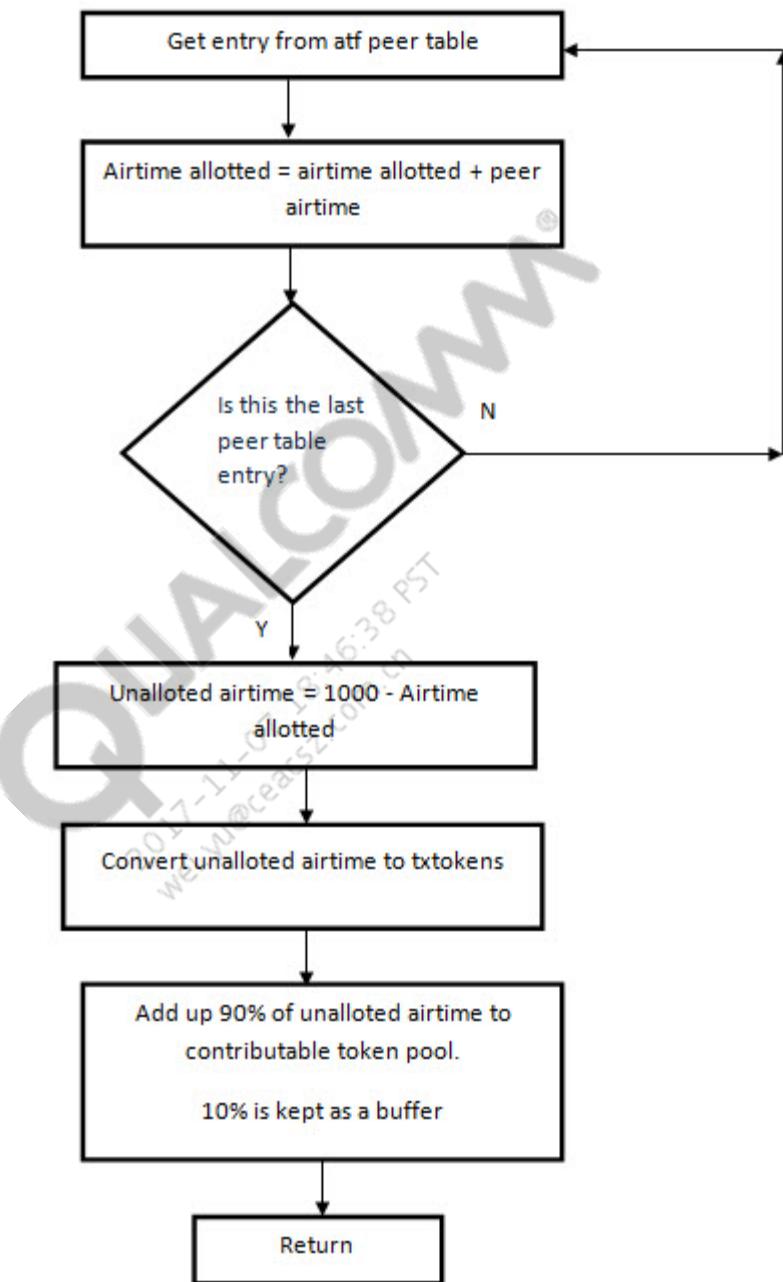
## Fair-queue algorithm flowchart



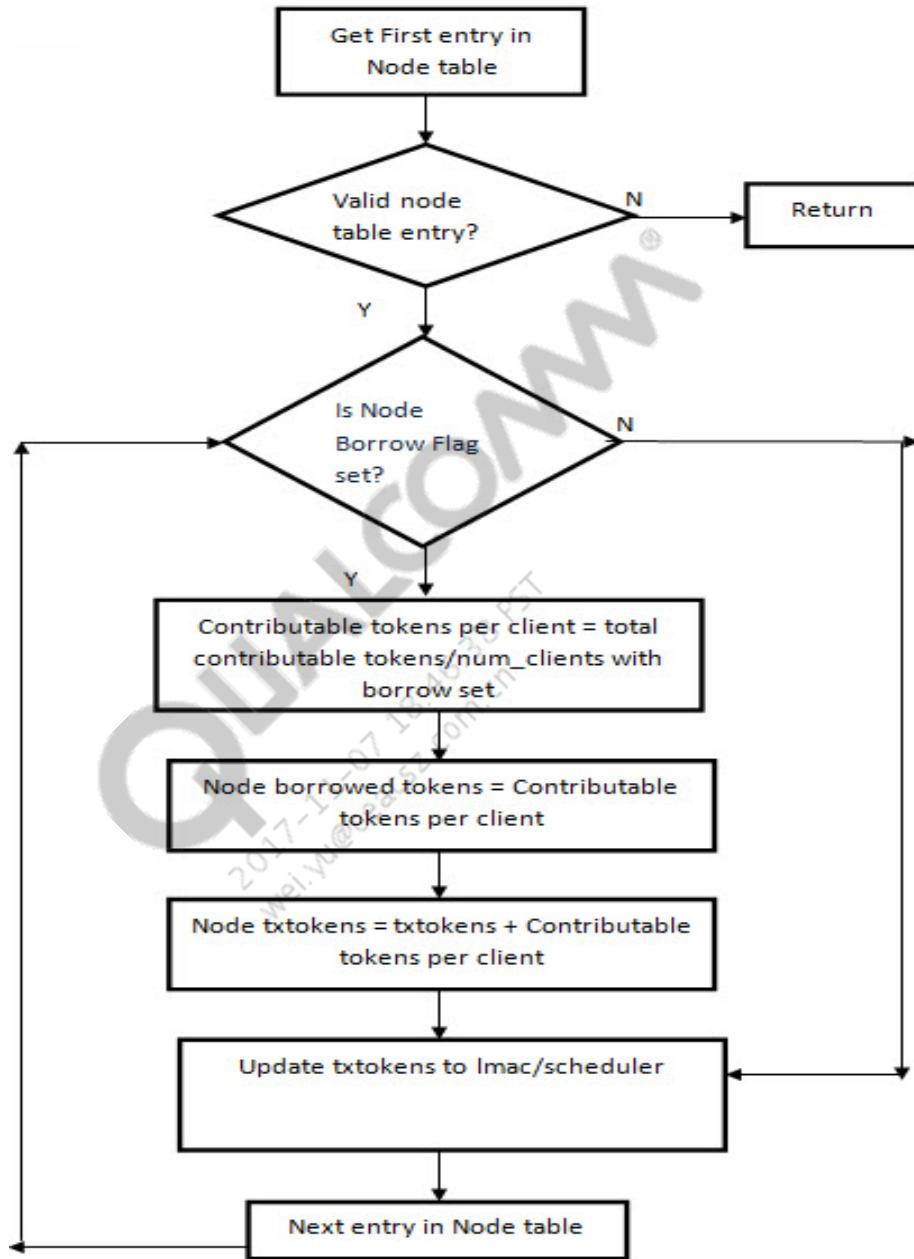
## Fair-queue algorithm



## Compute unallotted airtime



## Token distribution



### 10.6.3.4 ATF scheduler in direct attach

ATF scheduler changes for ATF can be sub-divided as follows:

- **Tx duration estimation:** Estimate Tx duration for the packet before transmit
- **Tx duration actual:** Calculate actual packet duration on transmit completion

## Tx duration estimation

The Tx duration estimate happens before each packet/aggregate transmit. Before adding the packet to the hardware queue, the packet duration is estimated by the routine `ath_pkt_duration`. Best MCS rate is considered for the duration estimate. The routine returns the duration in micro seconds. Transmission overheads including, SIFS, backoff and ack duration is added to the value returned by the routine.

The estimated duration is now compared with the tokens available for this node. If the estimated duration is less than the token available, the node token is updated (after deducing the estimated duration) and the buffer is send to the hardware for transmit. The estimated duration is also updated in the Tx descriptor. On the other hand, if the estimated duration is more than the tokens available, the packet is enqueued to the software queue and will be send out on the next token availability.

**NOTE** Node tokens are replineshes by the UMAC timer on every timer tick.

## Packet duration calculation on transmit completion

On receiving a transmit completion indication , the actual Tx duration for the packet is calculated. The MCS rate at which the packet was actually transmitted & the number of retries are read from the Tx descriptor. The actual Tx duration is then compared with the estimated duration already available in the descriptor.

If the actual duration is more than the estimated duration (in case of retries or transmit at a lower MCS rate), the difference is deducted from the node tokens in the next token replenish cycle. If the estimate duration is more than the actual duration, the difference is added to the node tokens in the next token replenish cycle.

### 10.6.4 ATF – OBSS Interference

Air Time distribution between the connected STAs can be accurately distributed, only if interference due to Other BSS are considered. The WLAN driver determines the actual available bandwidth, using the hardware cycle counters.

The following MAC PCU registers are available from the hardware:

1. MAC\_PCU\_RX\_CLEAR\_CNT- Rx Clear in Control channel
2. MAC\_PCU\_RX\_CLEAR\_DIFF\_CNT- Rx Clear in Extension channel
3. MAC\_PCU\_RX\_FRAME\_CNT- Rx Frame count
4. MAC\_PCU\_TX\_FRAME\_CNT- Tx Frame count
5. MAC\_PCU\_CYCLE\_CNT- Total Cycle count

Periodically, these counters are monitored and the channel availability is derived. This channel availability is used to accurately distribute the Tokens to the associated STAs.

## 10.6.5 ATF – Tx Buffer distribution

Apart from token distribution between different clients in the ratio of allotted airtime, we can also distribute the available Tx Buffers between these clients in the same ratio. This can be dynamically enabled/disabled using “iwpriv athx atf\_shr\_buf 1/0”. This is needed for protocols without flow control eg. UDP. Otherwise traffic flows for clients having less airtime can potentially hold majority of the Tx Buffers, leading to more packet drops for flows having larger airtime.

## 10.6.6 ATF—Supported maximum clients in direct attach

With ATF enabled, maximum clients supporting is limited up to 50. The limit is applicable for both offload and direct attach implementation.

On direct attach architecture, there is a separate 'maxclient' feature which can be enabled according the requirement. When this feature is enabled, maximum clients supported will be set to the default value (128). With this feature, 32 clients will be part of ATF table (ATF capable clients) and remaining clients outside ATF table will contend for any unassigned airtime. Note that this feature is disabled by default and is supported only in direct attach architecture. This feature can be enabled/disabled using the command 'iwpriv ath0 atfmaxclient 1/0'.

The 'maxclient' feature on direct attach solutions is explained in the following example:

Consider an AP configured with 1 VAP, VAP1. The VAP is assigned 80% of airtime through VAP based ATF configuration. If there are 100 clients connected to this AP, the first 32 clients will share 80% of airtime and the remaining 68 clients will contend for 20% of unassigned airtime.

**NOTE** whenever the number of clients supporting are changed, a VAP reset is initiated which in-turn triggers client disconnection.

## 10.6.7 ATF—SSID grouping support in direct attach

ATF implementation also supports SSID grouping, where groups can be created, with each group containing an SSID or multiple SSIDs. Airtime can then be assigned to a group which would be distributed within and across groups. The user has the option to select either strict or fair queue scheduling within the group & across the groups. Separate commands are available to configure both Intra-group & Inter-group scheduling policy.

## 10.6.8 Configure SSID grouping for ATF

This section explains the configuration of ATF SSID grouping functionality on the direct attach architecture. BSSIDs can be grouped together to allow a percentage of airtime to be shared across multiple BSSIDs. Transmit opportunities are allocated based on configurable weight (percentage of airtime). All clients in a BSSID group share the airtime percentage allocated to the BSSID group.

The following commands are added for ATF SSID grouping configuration:

1. **wlanconfig <vap name> addatfgroup <group name> <ssid>** : Creates a group (if it does not exist) and adds ssid to the group created. If the group already exists, adds the ssid to the group
2. **wlanconfig <vap name> configatfgroup <sairtime %>** : Reserve airtime for a group
3. **wlanconfig <vap name> delatfgroup <group name>**: Deletes a group
4. **wlanconfig <vap name> showatfgroup**: Displays the group configured.
5. **wlanconfig <vap name> showatftable**: Displays the group and client-wise airtime allocation.

To configure ATF SSID grouping and view the allocation, do the following:

1. Create three VAPs, namely, atf-lan1, atf-lan2, and atf-wan1
2. Create 2 groups, private and public, and add SSIDs to the group  

```
wlanconfig ath0 addatfgroup private atf-lan1
wlanconfig ath0 addatfgroup private atf-lan2
wlanconfig ath0 addatfgroup public atf-wan
```
3. Assign airtime to the groups created  

```
~ # wlanconfig ath0 configatfgroup private 80
~ # wlanconfig ath0 configatfgroup public 20
```
4. Enable atf configuration  

```
~ # iwpriv ath0 commitatf 1
```
5. Display the ATF group table:  

```
~ # wlanconfig ath0 showatfgroup
```

|         | SHOW | ATF               | GROUP |
|---------|------|-------------------|-------|
| Group   | ATF  | SSID LIST         |       |
| =====   | ==== | =====             |       |
| private | 800  | atf-lan1 atf-lan2 |       |
| public  | 200  | atf-wan           |       |

### 10.6.9 ATF—Dynamic enable/disable feature for direct attach architecture

On the direct attach radio, ATF feature can be enabled/disabled during run time using the following command:

```
iwpriv athX commitatf 1/0
```

**NOTE** The module parameter, atf\_mode is not applicable for direct attach architecture.

### 10.6.10 ATF—Guaranteed Throughput or Bandwidth Fairness

When a client connects, it first gets a share of the unallotted airtime.

At the next airtime allocation, list of all connected clients is traversed. For those which have a minimum throughput requirement, the achievable UDP throughput is calculated from the phryrate and PER. The throughput can be estimated assuming optimal aggregation and optimal packet size, that is already present in ‘userRateKbps’ field of the rate table. From that the airtime needed to satisfy the minimum throughput requirement is determined.

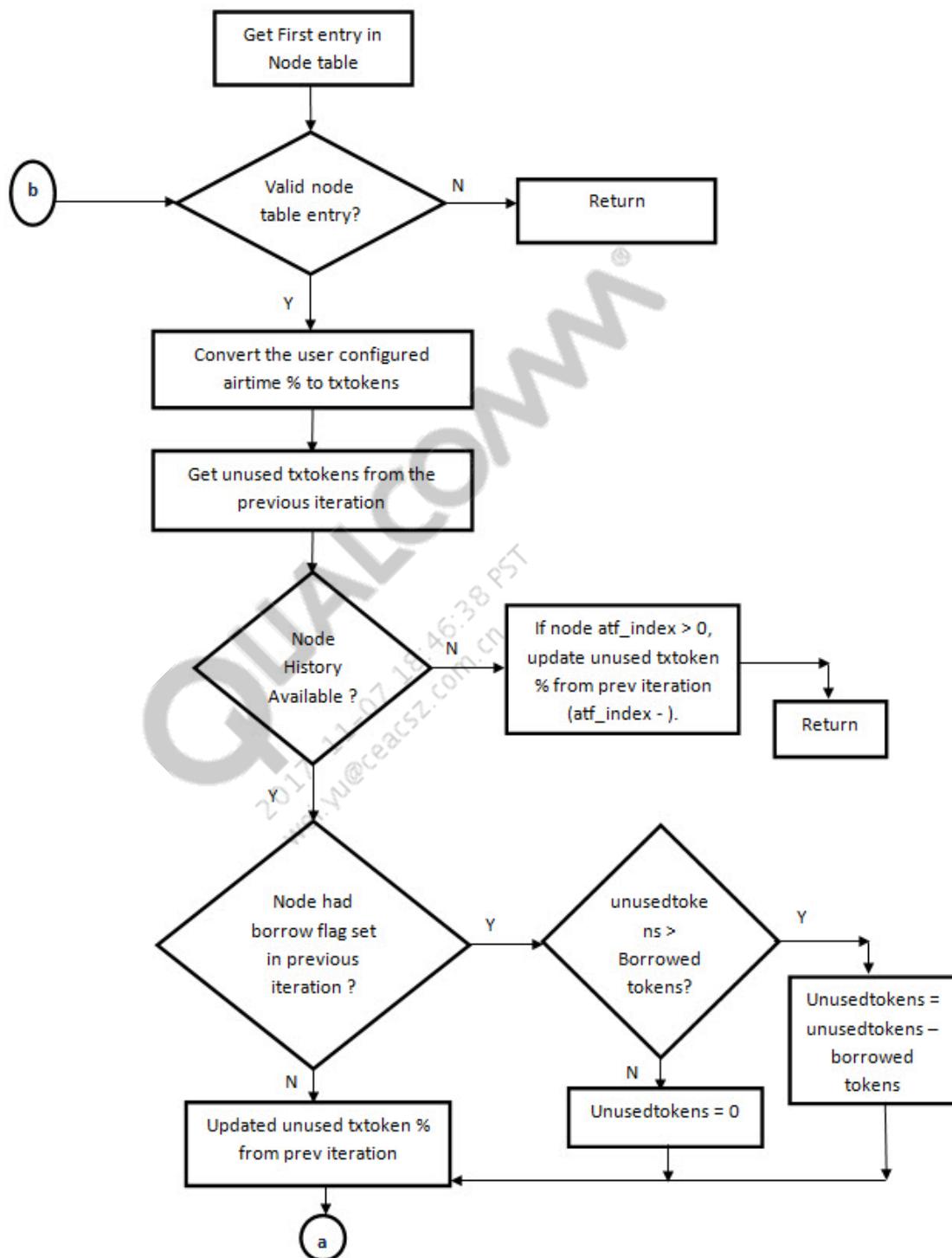
If the client had some airtime allotted to it, and the new airtime requirement is less than or equal to the allotted airtime, then it is configured with its new requirement. Else, the extra airtime needed for it is noted. If the client had some max airtime configured for it and the new requirement is more than that, then the new requirement is limited to the max airtime.

If there are no clients which have any extra requirement, the list of connected clients again is traversed again (from oldest to newest) to find out if there are any whose throughput requirement can now be met with the unallotted airtime available. Unallotted airtime is what was left after configuring the clients. If possible, such clients are configured with their airtime requirements and the unallotted airtime is adjusted accordingly.

Otherwise, the unallotted pool is used to fulfill the requirements of the clients which need extra airtime. But if still clients are left which need more airtime, and if the airtime requirement of such clients can be fulfilled by de-configuring other clients which are newer to them, then newer clients are de-configured to meet those airtime requirements. After this the list of connected clients is traversed again so see if the unallotted airtime can be used to fulfill the airtime requirement of any other client - this can happen if some clients got de-configured.

### 10.6.11 ATF—Per-SSID scheduling policy in Direct Attach

ATF implementation also supports per SSID scheduling policy configuration. The configuration change would be to mark/configure any SSID's that should follow strict scheduling policy. By default all SSID's will be configured to follow the fair scheduling policy. User would need to enter a command to configure a particular SSID in Strict mode. This configuration will come into effect only when radio level ATF scheduling policy is set to Fair.



## 10.7 Airtime configuration GUI

An intuitive and easily-navigable web interface is available to enable all of the ATF configuration options to be specified. Apart from the CLI interface that enables the ATF parameters to be configured, the web interface offers a simplified and optimal mode of administration of the ATF settings. The web or GUI interface enables the following ATF attributes to be defined:

- ATF for each SSID, such as 20% public or 80% private.
- Bandwidth allocation as a percentage of all clients associated with the AP or SSID. The interface shows the list of clients (active and inactive) and the user can assign a percentage to each AP or SSID. It is not necessary to use MAC addresses. Setting the bandwidth value as 0% indicates that the device has no specific allocation and must compete for residual time after the allocations are used.
- Strict and fair queue ATF algorithm selection on a per-SSID basis.
- Grouping of clients such as assigning clients to a group and allotting a time percentage for the group. Changes to allocations are immediately implemented without the need for a reboot or a restart of the AP.

The ATF configuration GUI is presented as a web page. Use a browser to navigate to the default address <http://192.168.X.X> (where X.X is dependent on integration) and enter the authentication credentials to log in to the GUI. Click the Airtime link on the left pane to access the ATF settings.

### Configure airtime fairness using the GUI

The Airtime configuration GUI contains two main components—Fairness and Performance. Click the Airtime link or icon on the left pane to view the Airtime configuration GUI. By default, the Airtime Fairness page is shown. To launch the Airtime Fairness page, click the Fairness link on the left pane. Click the Performance link on the left pane to view the Airtime Performance page. The ATF settings must be configured using the Airtime Fairness page before the performance statistics can be viewed using the Airtime Performance page.

To configure airtime fairness, do the following:

1. For the Enable Airtime Fairness field, click the On option button to enable airtime fairness. Alternatively, click the Off option button to disable airtime fairness. This is a toggle button. Airtime fairness must be enabled for the ATF parameters to be configured. When airtime fairness is enabled, the page refreshes to display fields to configure ATF settings.
2. For the Scheduling field, click the Fair option button to allow devices or MACs to fairly share unused airtime. Alternatively, click the Strict option button to enforce the limits and allow no sharing of the unused time. This is a toggle button.
3. For the WIFI - SSID section, do the following:
  - a. View the currently assigned airtime for the SSID in the current % field.
  - b. Click the Reserve field to view an option selector. Use the up and down arrows, or enter a value to specify the percentage of each second of airtime that the SSID is allowed to use.
  - c. Click the Defaults button to reset the SSID reservation to the default value.
4. Click the down arrow to the right of the WIFI – SSID section to expand and display the devices corresponding to the SSID.

5. For the Device – MAC section, do the following:
  - a. View the currently assigned airtime for the device in the Current % field.
  - b. Click the Reserve field to view an option selector. Use the up and down arrows, or enter a value to specify the percentage of SSID airtime that the device must be allowed to use.
  - c. Click the Default button to reset the SSID time for the device back to the default setting.
6. Complete the configuration of SSID time reservation for the different SSIDs and devices.
7. Click ‘Save & Apply’ to submit the configuration and for the changes to become effective for all the SSIDs and devices.
8. Click Cancel to reload the page and disregard changes.
9. Click Defaults to reset the SSID time reservation to the default values.

### Verify the configured airtime fairness on the console

Verify whether the ATF parameters configured using the GUI are applied correctly on the appropriate SSIDs and devices. To verify the configured ATF parameters, do the following:

1. After enabling or disabling airtime fairness on the Airtime Fairness GUI page, verify by entering the following command on the console:  

```
iwpriv ath0 get_commitatf
```

‘ath0 get\_commitatf:1’ denotes ATF is enabled.  
 ‘ath0 get\_commitatf:0’ denotes ATF is disabled.
2. After configuring strict or fair scheduling view on the Airtime Fairness GUI page, verify by entering the following command on the console.  

```
iwpriv wifi0 gatfstrictsched
```

Strict scheduling is denoted as ‘wifi0 gatfstrictsched:1’.  
 Fair scheduling is denoted as ‘wifi0 gatfstrictsched:0’.
3. In the WIFI – SSID section, after selecting a SSID’s reserve value for airtime and applying the set value or the default value, verify the following on the console:
  - a. Search for a line, such as iwconfig ath0 IEEE 802.11ng ESSID:"OpenWrt"
  - b. Find the SSID name, ‘OpenWRT’ in this case, and the interface of this SSID is ath0
  - c. With the interface name of the SSD identified, enter the wlanconfig ath0 showatftable command
  - d. View the output of the command:  

| SSID Client(MAC Address) | Air time(Percentage) | Config ATF(Percentage) |
|--------------------------|----------------------|------------------------|
| OpenWrt                  | 57.0                 | 57.0                   |

In the preceding output, the Config ATF (Percentage) field displays the SSID time reservation configuring using the GUI.
4. In the Device – MAC section, after selecting a device or MAC pertaining to an SSID, and specifying a reservation value or applying the default value, verify the following on the console:
  - a. Search for a line, such as iwconfig ath0 IEEE 802.11ng ESSID:"OpenWrt"

- b. Find the SSID name, ‘OpenWRT’ in this case, and the interface of this SSID is ath0
- c. With the interface name of the SSD identified, enter the following command.

```
wlanconfig ath0 showatftable
```

- d. View the output of the command:

| SSID              | Client(MAC Address) | Air time(Percentage) | Config ATF(Percentage) |
|-------------------|---------------------|----------------------|------------------------|
| OpenWrt           |                     | 57.0                 | 57.0                   |
| d8:50:e6:73:ef:b7 |                     | 17.1                 | 30.0                   |

- e. One of the rows beneath the SSID row in the output displays the MAC of the device that is changed. The Config ATF (Percentage) field of this row displays the value set using the GUI.

## Mapping of ATF GUI options and ATF CLI commands

This section describes the GUI options and the corresponding iwpriv commands available to configure ATF settings.

- Enable Airtime Fairness
  - On – Enable airtime fairness
 

```
iwpriv ath0 commitatf 1
```
  - Off – Disable airtime fairness
 

```
iwpriv ath0 commitatf 0
```
- Scheduling
  - Fair - Allow devices/MACs to fairly share unused airtime
 

```
iwpriv wifi0 atfstrictsched 0
```
  - Strict - Enforce the limits and allow no sharing of the unused time
 

```
iwpriv wifi0 atfstrictsched 1
```
- WIFI - SSID
  - Reserve – Percentage of each second of time is this SSID allowed to use
    - Use iwconfig to find the interface and ssid name to be changed, such as ath0 and OpenWrt
    - Set this SSID to 50% airtime by entering the following command.

```
wlanconfig ath0 addssid OpenWrt 50
```
  - Defaults – Reset the SSID reserve to default
    - Use iwconfig to find the interface and ssid name to be changed, such as ath0 and OpenWrt
    - Remove the SSID from the management by entering the following command.

```
wlanconfig ath0 delssid OpenWrt
```
- Device – MAC
  - Reserve – Percentage of the SSID time that the device is allowed to use

- Use iwconfig to find the interface and the SSID that this MAC address is on, such as ath0 and OpenWrt.
- Identify the MAC of the device that needs to be changed.
- For example, use the wlanconif command to view a list of all MACs connected.

```
wlanconfig ath0 showairtime
Client (MAC Address)           Air time (Percentage 1000)
d8:50:e6:73:ef:b7             171
70:14:a6:39:da:f7             141
```

- To specify the airtime reservation, enter the value for the corresponding MAC. For example, to specify 50% airtime for the MAC of d8:50:e6:73:ef:b7, enter as follows. The MAC address must be entered without the colons as separators.

```
wlanconfig ath0 addsta d850e673efb7 50
```

Default – Reset the SSID airtime reservation to the default setting

- Use iwconfig to find the interface and the MAC address that corresponds to the SSID, such as ath0 and OpenWrt.
- Identify the MAC of the device that needs to be changed. For example, use the wlanconif command to view get a list of all macs connected

```
wlanconfig ath0 showairtime
Client (MAC Address)           Air time (Percentage 1000)
d8:50:e6:73:ef:b7             171
70:14:a6:39:da:f7             141
```

- To specify the airtime reservation, enter the value for the corresponding MAC. For example, to specify the default airtime for the MAC of d8:50:e6:73:ef:b7, enter as follows. The MAC address must be entered without the colons as separators.

```
wlanconfig ath0 addsta d850e673efb7
```

## View the ATF performance using the GUI

To launch the Airtime Fairness page from the Airtime configuration GUI, click the Fairness link on the left pane. Click the Performance link on the left pane to view the Airtime Performance page. The ATF settings must be configured using the Airtime Fairness page before the performance statistics can be viewed using the Airtime Performance page.

The Airtime Performance page presents a live and most-recent display of changes made to the percentage of radio time used globally, per SSID, and per MAC address. The values are displayed in the form of a line graph with time in intervals of 30 seconds displayed along the x-axis and the percentage of radio time displayed along the y-axis.

To view the ATF performance, do the following:

1. In the Global Airtime Fairness line graph at the top of the page, view the values plotted over the last 60 seconds of global airtime percentage for all the SSIDs. The time intervals are shown along the x-axis and the airtime percentage is shown along the y-axis.
2. For the WIFI - SSID section, do the following:

- a. View the currently assigned airtime for the SSID in the Reserve % field. The band or meter displayed shows airtime percent in blue, overage percent in orange, and reserve percent in gray.
- b. These values correspond to the consolidated metrics for all the devices or MACs associated with the SSID.
3. Click the down arrow to the right of the WIFI – SSID section to expand and display the devices corresponding to the SSID.
4. For the Device – MAC section, do the following:
  - a. View the currently assigned airtime for the SSID in the Reserve % field. The band or meter displayed shows airtime percent in blue, overage percent in orange, and reserve percent in gray.
  - b. These values correspond to the metric for each device or MAC associated with the SSID.

### Verify the ATF performance GUI values on the console

Verify whether the ATF parameters displayed on the GUI match accurately with the ATF settings displayed on the console. To verify the configured ATF parameters, do the following:

To obtain the reserve and current percent of airtime for an SSID:

- Use iwconfig to find the interface and ssid name you want to view, such as ath0 and OpenWrt respectively.
  - Run the ‘wlanconfig ath0 showatftable’ command and view the output.
- | SSID Client(MAC Address) | Air time(Percentage) | Config ATF(Percentage) |
|--------------------------|----------------------|------------------------|
| OpenWrt                  | 57.0                 | 57.0                   |
| d8:50:e6:73:ef:b7        | 17.1                 | 30.0                   |
- The first row in the preceding output with OpenWRT as the SSID name shows the current airtime under the Air time(Percentage) field, and the reserve airtime under the Config ATF(Percentage) field.
  - The MACs that are not in the list have a reserve of 0% and are unmanaged devices.

To obtain the reserve and current percent of airtime for a Device – MAC (managed):

- Use iwconfig to find the interface and ssid name you want to view, such as ath0 and OpenWrt respectively.
  - Run the ‘wlanconfig ath0 showatftable’ command and view the output.
- | SSID Client(MAC Address) | Air time(Percentage) | Config ATF(Percentage) |
|--------------------------|----------------------|------------------------|
| OpenWrt                  | 57.0                 | 57.0                   |
| d8:50:e6:73:ef:b7        | 17.1                 | 30.0                   |
- The second row in the preceding output with d8:50:e6:73:ef:b7 as the client MAC address shows the current airtime under the Air time(Percentage) field, and the reserve airtime under the Config ATF(Percentage) field.

To obtain the reserve and current percent of airtime for a Device – MAC (managed):

- Use iwconfig to find the interface and ssid name you want to view, such as ath0 and OpenWrt respectively.

- Run the ‘wlanconfig ath0 list sta’ command and view the output.

| ADDR              | CAPS | ACAPS | AID | CHAN              | TXRATE | RXRATE | RSSI | IDLE | TXSEQ | RXSEQ |
|-------------------|------|-------|-----|-------------------|--------|--------|------|------|-------|-------|
| 17746             | 6112 | ESS   | 0   | d8:50:e6:73:ef:b7 | 1      | 11     | 0M   | 0    | 6M    | 22    |
| 70:14:a6:39:da:f7 | 0    | 1f    | 3   | 11                | 0M     | 38M    | 28   | 0    | 316   | 15    |
| 0                 | 1f   | 0     |     |                   |        |        |      |      | 5584  | ESS   |

The MAC address of each device is listed here for both managed and unmanaged devices.

## 10.8 ATF event logging

The ATF event logging functionality enables you to view the airtime used by a client as a percentage of total airtime allotted across all clients. This is updated every ATF interval. Also, the total airtime used by a client from the time of connection of the client. The client state transitions events – getting connected / dis-connected and the static airtime allocation for each client are logged.

Airtime used does not vary linearly with traffic because of effects of aggregation. Enabling the logs causes a slight degradation in the throughput.

Enable or disable logging of airtime used as a percentage of total airtime allotted across all clients, at every ATF interval, using the iwpriv wifiX ath\_log <0 | 1> command.

Example Command:

```
#iwpriv wifi0 ath_log 1
/* enable ATF logging */
#iwpriv wifi0 ath_log 0
/* disable ATF logging */
```

For direct attach radios, at every ATF token allocation interval, the airtime used by a client is calculated as a percentage of total allotted airtime (across all clients). Also, the airtime that is used by a client since the time the client is connected is calculated. These values are calculated for both strict and fair scheduling. These two statistics can be queried using wlanconfig command. Also, at every ATF token interval, the airtime used by a client as a percentage of total allotted airtime (across all clients) is logged. However, this logging is enabled through an iwpriv, as it degrades performance. Client connection and disconnection events and the static airtime allocations for each client are also logged.

For offload radios, a WMI command is used to enable the firmware to send used and unused airtime for every client at every ATF token allocation interval. This is sent to host using WMI event. With the used and unused airtime computed, the host calculates the airtime used by a client as a percentage of total allotted airtime (across all clients), and we also calculate the airtime that is used by a client since it got connected. This computation is performed for both strict and fair scheduling. These two statistics can be queried using the wlanconfig athX atfstat <mac address> command. Also, at every ATF token interval, the airtime used by a client as a percentage of total allotted airtime (across all clients) is logged. However, this logging is enabled through an iwpriv, as it degrades performance. Client connection and disconnection events and the static airtime allocations for each client are also logged.

## 10.9 ATF fair-scheduling based on WMM access classes

A functionality to enable ATF fair-scheduling based on WMM classes is introduced. This feature takes allocated airtime for a peer into consideration, regardless of QoS traffic type that it runs. This feature restricts the peer only to use the allocated airtime irrespective of high priority or low priority traffic, allowing peer to transmit only allocated airtime of data. This feature works when AP operates in ATF mode. Amount of data pumped is more than expected TPUT for any type of traffic. About 10 % variance of throughput is considered to be acceptable when using this feature.

Consider a scenario in which two peers A and B configured with 80:20 airtime, where peer A run BE traffic and peer B runs VI traffic. When peer B exhausts its tokens, it must not occupy any airtime until peer B exhausts all its available tokens. Such a behavior enables the achieved throughput to be divided in the ratio of 80:20. Each WMM AC has its own scheduler context that dispatches schedule commands upon availability of data in queued TID. Freeloaders are scheduled when TID specific scheduler command queue remains free for three consecutive schedule intervals. A mechanism is introduced to prevent fair-scheduling until all the peers that has traffic has exhausted the tokens. Before allowing free loaders or peers with no ATF tokens for scheduling, it is ensured that all other ATF peers have exhausted their tokens.

## 10.10 ATF bandwidth fairness or minimum guaranteed throughput

The ATF bandwidth fairness feature allows network administrator to specify a minimum guaranteed bandwidth/throughput for a peer which shall be accommodated by WLAN firmware by allocating appropriate airtime. Peers are prioritized based on the order of association, peer associated first gets more priority.

ATF bandwidth fairness feature is designed based on sampling, data rate and amount of bytes transmitted to a peer is sampled for a fixed ATF cycle duration of one second. Based on this sample data, airtime required for a peer to achieve the configured throughput in the next ATF cycle is determined by the firmware. When all the configured peer throughputs are achieved remaining airtime would be shared among all the peers thus utilizing the complete network capable bandwidth.

Keep the following points in mind while using the ATF bandwidth fairness feature:

- Configuration for throughput to be done based on per peer MAC only.
- This feature supports up to 10 throughput configured peers max.
- This feature works in ATF Fair mode, since we assure minimum guaranteed throughput.
- This feature is designed to use max network bandwidth, however improper configuration may lead to reduction in cumulative throughput.
- More data should be pumped via Ethernet than the expected throughput; for example, Expecting 250 Mbps from a TPUT peer requires at least 400 Mbps to be pumped from Ethernet.
- AP switches to ATF throughput mode for supporting this feature, which is exclusive from normal ATF mode.

- The throughput specified by the users shall be within the Best Effort UDP / TCP throughput attained with the same set of peers without throughput mode enabled.
- The oldest client is the first associated client. From Firmware perspective, this always get the maximum priority.
- Some percentage of airtime (default 10%) will be always reserved for non-configured clients and this reserved airtime is configurable.
- Throughput configured peers requires two mandatory arguments throughput in kbps and maximum airtime which shall be allocated for the peer in %.
- Throughputs configured should be more than 25 Mbps i.e., 25000 Kbps.
- Peers will be prioritized based on the order of association.
- Peer which has higher priority shall get about 20% of total bandwidth, when the peer has less throughput when compared to lower priority peers. Hence ideal way is to sort the priority of peer in decreasing order of the throughput. However any type of configuration is supported in this feature.
- With throughput fairness, we can expect 10% degradation of throughput to what is configured on non-ideal conditions. For example, for a 50 mbps configuration, the achievable shall be around 45. The deviation could go even higher if the client behavior is abnormal or buggy.
- Once this feature is enabled, to see the appropriate bandwidth accordingly, it might take few seconds to stabilize. Since, we depend on the past TX history for the given client to allocate airtime.
- Enabling or disabling this feature might lead to clients getting disconnecting and connecting back (depends on client's implementation on how it handles retry scenarios or how it handles sudden drops in throughput).
- The time taken for throughput to stabilize after a client has moved closer or farther will depend to an extent on the upper layer protocol and the traffic generating application. For example, the UDP throughput will stabilize faster than the TCP throughput.
- This feature will assign airtimes to different clients based on per second rate and volume transmitted, so when this feature is enabled user should not override the airtime allocations from any command line tools during test scenarios.

## 10.11 ATF strict-scheduling per SSID or virtual device for QCA9880

In addition to the support for specifying ATF strict scheduling per radio, the capability to specify ATF strict scheduling per SSID or virtual device is introduced.

The vdev parameter to set and clear flag for ATF strict scheduling for that SSID or virtual device (vdev) is implemented. When more than one peer is connected to a vdev, the vdev airtime will be equally divided among them. When this vdev is marked strict, each peer's airtime will be maximum (vdev airtime/num peer for vdev). This is not applicable when strict scheduling is set for the entire radio (global policy set to strict mode)

Whenever user configures strict scheduling global policy, per SSID configuration is not reset. While changing to fair sched global policy, the previously configured per ssid strict scheduling takes effect.

The following are the infrastructural changes:

- Added flag in wal\_vdev sturct for strict sched
- Perform count of strict sched vdev in wal\_pdev
- Handle the vdev param sent by host in wlan\_vdev.c and set the flag appropriately in wal\_vdev

The following are the ATF-related changes:

- While allocating/redistributing tokens, in addition to the check for global strict sched, a check is introduced to determine whether the peer's vdev is marked strict sched. If vdev marked strict, tokens are allocated to that peer in the same manner in which it is done during global strict policy.
- Similarly, extra tokens are not allocated to peers connected to strict sched vdev.

The following are the scheduler changes:

- When the peer (strict vdev) runs out of token, its packets are queued in waitq.
- Whenever the waitq is triggered, a check is performed if the peer is connected to strict vdev or not.
- If connected to strict vdev, remove the packet from head and again queue it at the end of waitq. The particular packet is not sent in air.
- During the next token distribution, packets of the peer concerned from waitq are moved to runq and sent out based on the available token.

Use the following command to set ATF strict scheduling per SSID.

```
iwpriv athN atfssidsched {1/0}
```

## 10.12 ATF strict scheduling per SSID or virtual device for QCA9984, IPQ4019, and QCA9886

On QCA9984, IPQ4019, and QCA9886 chipsets, in addition to the support for specifying ATF strict scheduling per radio, the capability to specify ATF strict and fair scheduling per SSID or virtual device is introduced.

The vdev parameter to set and clear flag for ATF strict scheduling for that SSID or virtual device (vdev) is implemented. When more than one peer is connected to a vdev, the vdev airtime is equally divided among them. When this vdev is marked strict, each peer's airtime is maximum (vdev airtime/num peer for vdev). This behavior is not applicable when strict scheduling is set for the entire radio (global policy set to strict mode).

Whenever a user configures strict scheduling global policy, per-SSID configuration is not reset. Packet for a peer with no airtime can be transmitted as part of another peer's airtime, when the device is run in multiuser-multiple input multiple output MU-MIMO. All other existing limitations

of ATF & ATF-MU-MIMO apply. The ATF strict and fair scheduling per SSID functionality is supported for QCA9980 chipsets.

Use the `iwpriv athN atfssidsched {1/0}` command to enable or disable ATF strict scheduling per SSID.

## 10.13 ATF support for MU-MIMO

With multi-user multiple input multiple output (MU MIMO) and ATF enabled, the feature guarantees that each client receives airtime that is equal to or more than the airtime than the client receives with single user (SU) and ATF enabled.

A combination of MU and ATF provides equal to or better throughput for the same allocations done in the combination of SU and ATF modes.

MU peers do not negatively impact SU peers in ATF mode. SU peers must receive equal throughput with or without MU peers in ATF mode for the same allocation.

ATF + MU MIMO fair scheduling will be supported. STA with no tokens will be scheduled if STAs with tokens have no data to be sent. This will be transmissions on its own or along with STAs with tokens. Here, a token refers to a particular airtime percentage.

ATF + MU MIMO strict scheduling will be supported. To improve medium utilization, STAs with no tokens will still be transmitted, with STAs with tokens. It is possible for a user slot to be available in the MU PPDU, and for all other STAs to have tokens and not have data to transmit. Here, token refers to a particular airtime percentage.

In fair scheduling, transmission is allowed although all users in MU PPDU has zero tokens. In strict scheduling, transmission is not allowed if all users in MU PPDU have zero tokens.

To enable support for ATF for MU-MIMO, use the same commands as the ones used for ATF for single user (SU). MU capability in wireless configuration file must not be disabled.

To support SSID level percentage for MU-MIMO, use the following command:

```
wlanconfig athX addssid <ssid><allocation percentage>
iwpriv <interface> commitatf 1
```

To support mac level percentage, use the following command:

```
wlanconfig athX addsta <mac without colon><allocation percentage>
iwpriv <interface> commitatf 1
```

## 10.14 ATF hierarchy distribution

Airtime management (ATM) or airtime fairness (ATF) settings are valid in a hierarchy with the following order or priority in the hierarchical arrangement: radio, SSID, and client MAC. When ATF settings are configured for a client MAC, the configuration is valid only when the client MAC or peer MAC is within the specific radio and SSID.

When a peer MAC is configured with airtime percentage, the ATF percentage applies for that SSID only. If the peer MAC is already configured with ATF percentage for the SSIDs between which it switches, the ATF setting configured for those SSIDs applies. If the peer MAC is not configured with ATF percentage when it switches between SSIDs, then it shares the ATF percentage in the SSID to which it moves, based on the available airtime present in that SSID. When the peer MAC moves back to previous SSID, the previously allocated airtime for the peer MAC in that SSID applies.

Before the implementation of this functionality, peer MAC is allocated airtime from the global ATF percentage. Consider the following network topology with a 2 GHz band that has two SSIDs—HomeSSID with 40% ATM allocation and PublicSSID with 10% ATM allocation. HomeLaptop is allocated 70% of airtime in HomeSSID (that is,  $70\% \times 40\% = 28\%$  of overall 2G airtime).

When this HomeLaptop moves to PublicSSID, it shares airtime with other clients in 10% of 2G PublicSSID allocation if total airtime for all clients  $\leq 10\%$ ; else, reassignment of airtime is needed. HomeLaptop has  $70\% \times 10\% = 7\%$  of overall 2G airtime, assuming no other client is in PublicSSID. However, if HomeLaptop requested 70% and PublicSSID has only 3% remaining, it prompts for reassignment and assigns 0% ATF to the client that moved to the PublicSSID. When the HomeLaptop moves back to HomeSSID, it resumes 28% allocation, assuming no reassignment is configured; else, the changed airtime reassignment value applies in the HomeSSID.

With the implementation of this functionality, peer MAC is allocated airtime within the hierarchy of the radio and SSID. Consider the following network topology with a 2 GHz band that has two SSIDs—HomeSSID with 40% ATM allocation and PublicSSID with 10% ATM allocation. HomeLaptop is allocated 70% of airtime in HomeSSID (that is,  $70\% \times 40\% = 28\%$  of overall 2G airtime). HomeLaptop is allocated 30% of airtime in PublicSSID that is,  $30\% \times 10\% = 3\%$  of overall 2G.

When this HomeLaptop moves to PublicSSID, it shares airtime with other clients in 10% of 2G PublicSSID allocation based on the configured value. HomeLaptop has  $30\% \times 10\% = 3\%$  of overall 2G airtime, assuming no other client is in PublicSSID. However, clients might have already used more than 7% ATF that the user configured in PublicSSID. In such a case, user is prompted for reassignment. HomeLaptop is not configured for PublicSSID; instead, it takes the same percentage as configured for HomeSSID to maintain backward compatibility.

Instead of providing global ATF percentage, the changed ATF algorithm sends the VAP to peer MAC airtime to the lower layers for further processing. The notification that the throughput requirement of a client cannot be satisfied is sent, either when the client attempts to connect or when it is already connected.

### 10.14.1 Guidelines for ATF hierarchy distribution

Keep the following points in mind while employing the ATF hierarchy functionality:

- Peer MAC is configured with airtime per SSID or SSID is not considered if airtime is configured for the same VAP.
- Based on the available airtime in an SSID, the configured airtime is allocated to the peer MAC.
- If the configured airtime is not available, the user is prompted to reassign the airtime.

- After the peer moves to a new SSID, the value of airtime in that SSID is determined and peer is configured with that airtime percentage if it is available. Otherwise, the user is notified to reassign ATF when sufficient airtime is not available.
- When the peer MAC connects to previous SSID again, the corresponding value of airtime in the earlier SSID is taken into consideration and configured.
- If the user does not provide any SSID, only the VAP on which ATF is set uses the specified. When the VAP moves to other SSIDs, the same percentage is used to maintain backward compatibility.
- While deleting configuration, user can delete per-SSID based configuration or configuration for all SSID at a time. Enter the *wlanconfig athx delsta mac* command to delete the station globally, or enter the *wlanconfig athx delsta mac<sta mac addr>* command to delete the specified station.
- Enter the *wlanconfig athX addsta mac per* command to assign percentage of airtime globally. Enter the *wlanconfig athX addsta mac per ssid1* command to assign percentage of airtime per SSID.
- Enter the *wlanconfig athX delsta mac per* command to delete percentage of airtime globally.
- Enter the *wlanconfig athx showatftable [index:-to show per\_peer\_table]* command to display the ATF percentage configured for a particular peer in a particular SSID. An index value of 0 is a global percentage. Indexes starting from 1 are for corresponding VAP IDs.

## 10.15 Scatter/Gather DMA

The AR900B hardware supports scatter/gather DMA. For any DMA transfer, the first issue to consider is that the user may request a large transfer (kilobytes to megabytes) to a given buffer. This area may be contiguous in the virtual space, but can be composed of a sequence of pages fragmented all over physical memory. Features like TSO which process extra large frames work only when the scatter/gather DMA is supported in the hardware.

### 10.15.1 Implementation

Each packet transmitted is checked whether it is a linear packet or a non-linear packet with the “*qdf\_nbuf\_is\_nonlinear(skb)*” inline function; if it is a non-linear packet the “*ol\_tx\_sg\_process\_skb*” API is called for further processing.

The “*ol\_tx\_sg\_process\_skb*” API iterates through the fragment list of the skb and saves the fragment address and length in a software scatter/gather descriptor “*struct ol\_tx\_sg\_desc\_t*”. A pointer to this descriptor is saved in the cb array of the skb which is later used to fill the “*htt\_frag\_desc*” descriptor, so that the hardware can DMA from these fragment addresses.

### 10.15.2 Configuration

The ethtool utility is used to enable or disable the sg feature.

```
ethtool -K ath0 tx on/off
ethtool -K ath0 sg on/off
```

The iwpriv tool is used to get and reset the sg statistics.

```
iwpriv ath0 get_sg_stats
iwpriv ath0 rst_sg_stats
```

## 10.16 Coordinated ATF between root AP and repeater

This feature enables the user to control/configure ATF settings of the repeater AP from the root AP. The user must be able to configure the following ATF settings:

- SSID-based ATF configuration—addssid and delssid
- STA-based ATF configuration—addsta and delsta
- Group-based ATF configuration—addgroup and delgroup

The user enters ATF configurations for a repeater including the repeater MAC address at the root AP, which in turn is forwarded to the repeater AP and configurations are applied. The coordinated ATF feature uses wsplcd daemon to achieve this behavior. The configuration details are available in the *Wireless LAN Access Point (Driver Version 10.4) Command Reference Guide* (80-Y8052-1).

## 10.17 Peer flow control data path

Transmit data path has been changed to be peer-based flow control for better Transmit performance (Downlink) in scenarios like MU-MIMO multi-client, large number of SU clients, ATF, WMM and so on.

The WLAN target subsystem works with a limited amount of allocated memory and it allocates a limited set of descriptors in the transmit data path. Generally, this set of descriptors is tuned for peak performance of the WLAN Phy. But this limited set of descriptors is not optimally used when certain use-cases are exercised.

The following are the list of use cases that requires efficient management of the descriptor resources and the new implementation solves this problem by using the host processor and memory to accumulate the backlog while the firmware processor schedules and pulls in descriptors to its limited pool in order to maximize the throughput and the user experience.

The new implementation provides solution for following list of use-cases:

- MU-MIMO performance with large number of clients – In the original design, since the MU grouping algorithms run in the firmware and the host downloads the descriptors to the target, that may not be correlated to the TxOP of the next MU group that the firmware scheduler decides, there is lack of optimality of descriptor utilization in the target.
- Weak sister problem – When there are multiple data flows to different peers with different PHY rate capabilities, the limited descriptors in the firmware are consumed by a peer that has a lower data rate (for example, 802.11n client) thereby hogging another peer that has a higher data rate.
- Multi client performance – The aggregate throughput of the system with multiple peers should not degrade sharply as the number of active clients gradually increases.

- Air Time Fairness (ATF) – ATF algorithm run in the firmware and the host downloads descriptors to target for unrelated clients whose TXOP may not be scheduled by the firmware.
- Video over wireless
- WMM
- Peer caching with large number of active clients

### 10.17.1 Packet flow

In the new peer-based flow schedule design, host inspects packets coming in from the network stack/NSS Driver/Wi-Fi-VAP-VAP forwarding module, and classifies the packets and stores them in per-peer/TID queues instead of sending. For unicast frames, peer lookup is done based on destination MAC address. Multicast/broadcast frames are assigned to AP-BSS peer queue. TID is extracted from DSCP/QoS information in 802.11 QoS/ IPv4/v6 headers.

## 10.18 Content-aware routing

### Direct Attach

For Direct Attach, the wbuf\_classify() routine checks the TOS field of the IP header to determine the WMM priority.

When this feature is enabled, and the skb->queue\_mapping is non-zero, the wbuf\_classify() routine uses this instead of the TOS field to determine the WMM priority.

### Offload

For Offload, the ol\_tx\_classify() routine checks the TOS field of the IP header to determine the WMM priority, when classification on host is done. When this feature is enabled, and the skb->queue\_mapping is non-zero, the ol\_tx\_classify() routine will use this instead of the TOS field.

When host is not classifying, then it passes HTT\_TX\_EXT\_TID\_INVALID to the firmware. When this feature is enabled, and the skb->queue\_mapping is non-zero, a valid tid is passed to the firmware based on the WMM priority indicated by skb->queue\_mapping.

## 10.19 Override MU-MIMO capability based on vendor OUI

Phones become 2x2 spatial streams-capable only when such phones work with APs that are not MU-Capable. The workaround (WAR) implements a hack where we ensure that S7 phones are in SU-2x2 when it is the only MU-Capable client in the BSS. Also, all phones are ensured to be in MU-1x1 mode when other MU-Capable clients are present in the BSS (including presence of more than 1 MU-1x1 phone). When the MU-CAP-WAR is enabled, the Probe-Response is modified before sending to dedicated-client so that it joins as SU-2x2

The MU-Disabled Probe-Response (MU-Disabled in VHT CAP) is sent to the clients that contain the Broadcom Vendor-IE. This hacked probe response is sent only when the following conditions are satisfied:

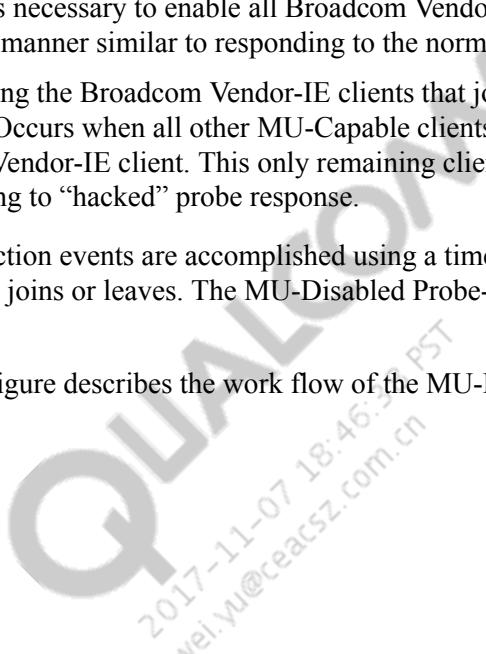
- The number of MU-Capable clients in the system is 0.
- The override variable is not set (this override variable prevents certain race conditions).

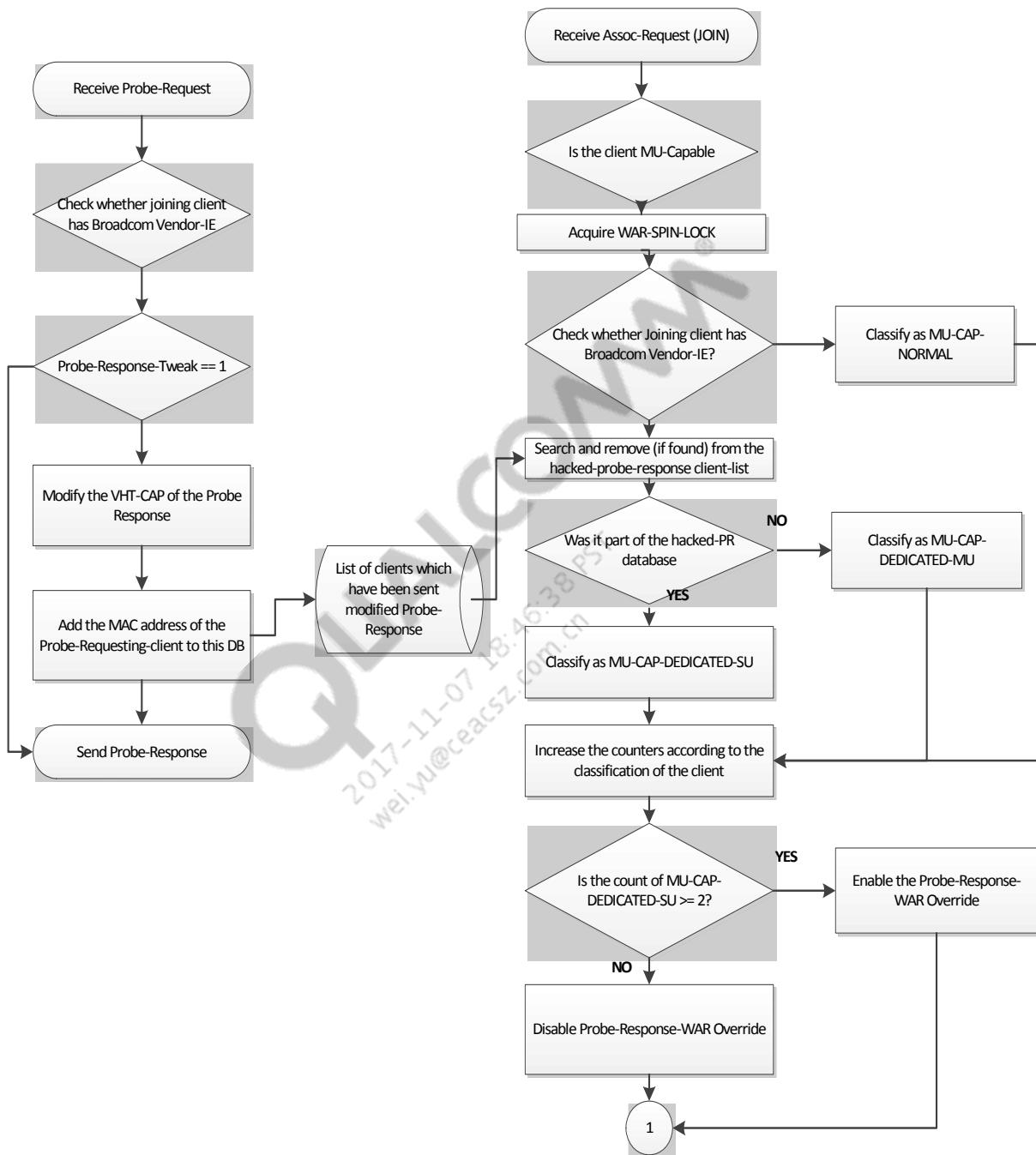
In addition, the associated clients with Broadcom Vendor-IE are kicked out with a deauthentication frame, so that they can rejoin as per the desired state (MU-1X1 or SU-2x2). This kick-out is triggered with a 15-second delay, when any MU-Capable client (including any Broadcom Vendor-IE client) joins or leaves the system. There are 2 kinds of kick-outs

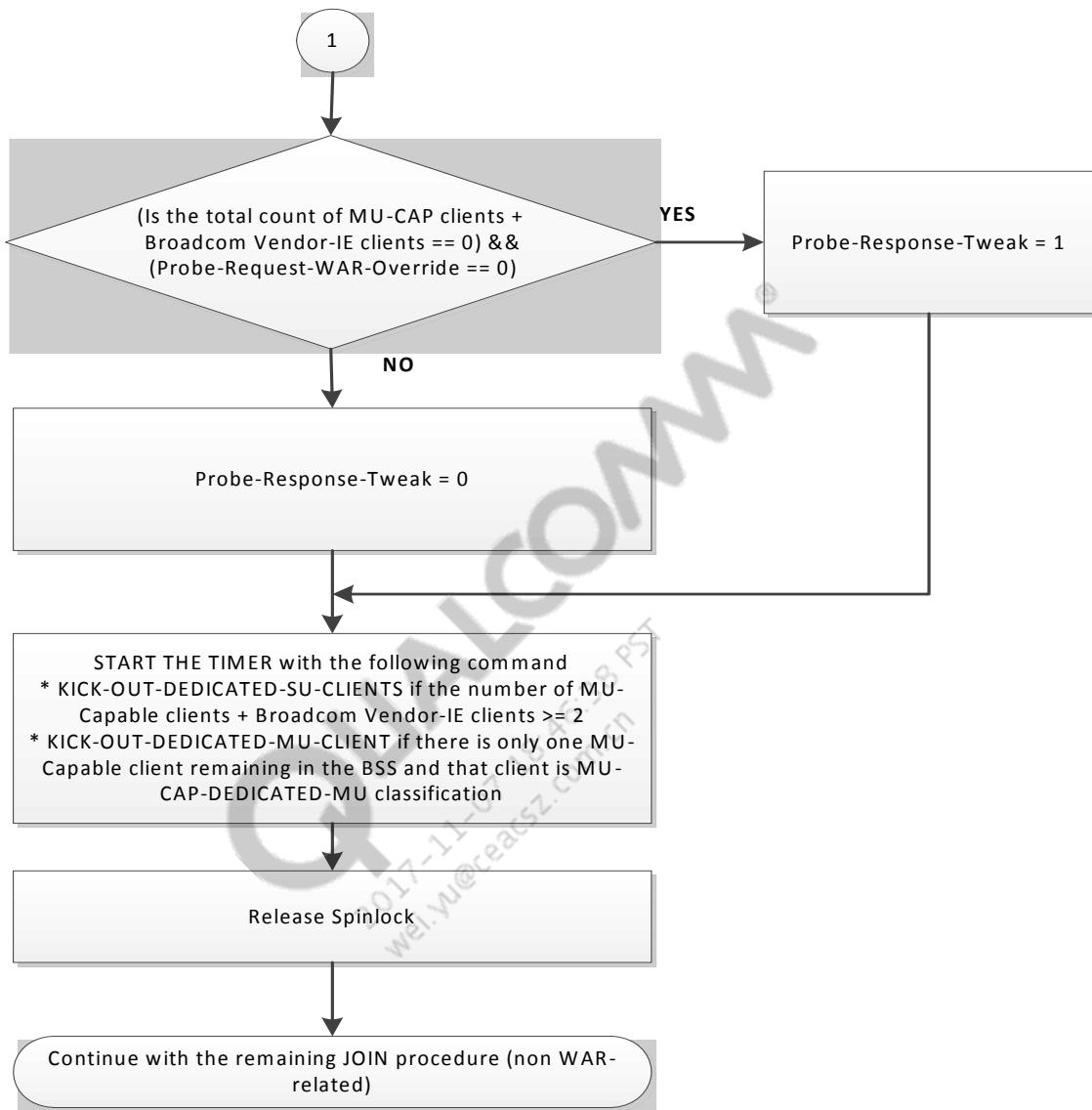
- Disconnecting the Broadcom Vendor-IE clients that joined as a result of Hacked-Probe Response---Occurs when other MU-Capable (including other Broadcom Vendor-IE) clients join and it is necessary to enable all Broadcom Vendor-IE clients join “normally” (that is, joining in a manner similar to responding to the normal probe-response).
- Disconnecting the Broadcom Vendor-IE clients that joined as a result of normal Probe response---Occurs when all other MU-Capable clients leave except the single remaining Broadcom Vendor-IE client. This only remaining client is disconnected and it is made to join as responding to “hacked” probe response.

These disconnection events are accomplished using a timer which kicks in after 15 seconds from the time a client joins or leaves. The MU-Disabled Probe-Response is referred to as “hacked” probe response.

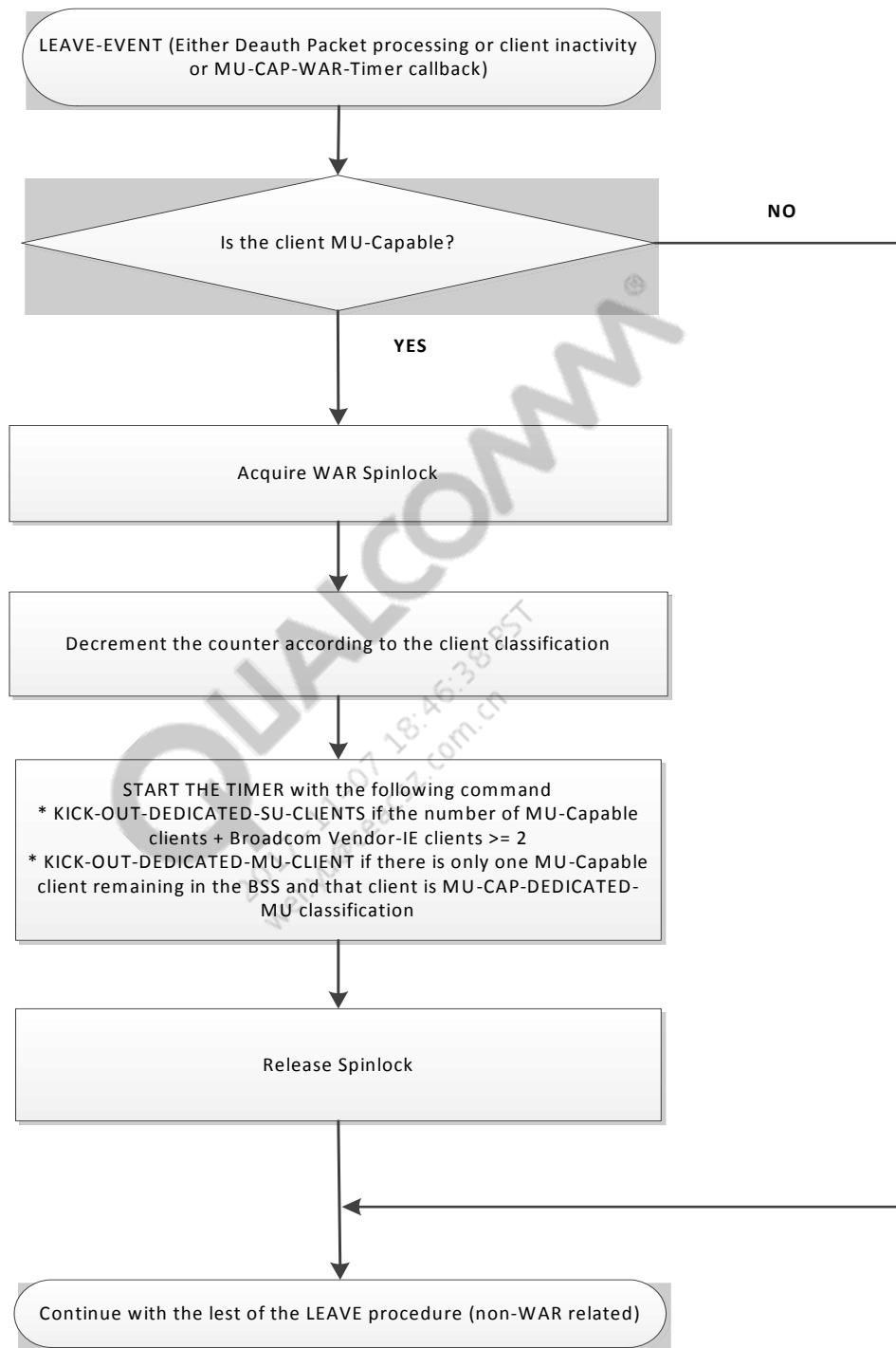
The following figure describes the work flow of the MU-Disabled Probe-Response:



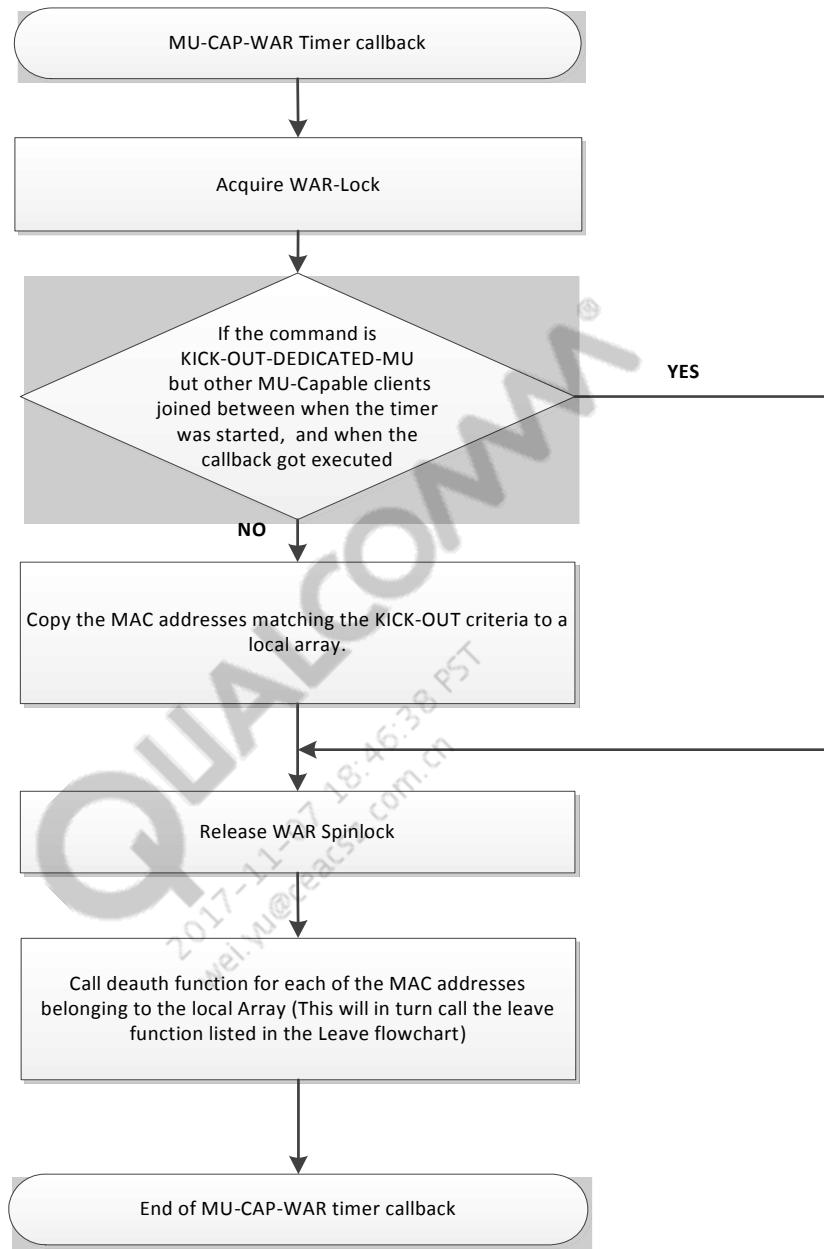




The following figure describes the workflow of the WAR spinlock:



The following figure describes the workflow of the WAR spinlock:



The following workflow illustrates the events that occur when an MU-capable client joins and with the WAR feature (MU-MIMO override capability based on vendor OUI) is enabled or disabled:

- Enable VAP-level debugging logs.  
`iwpriv ath0 dbgLVL 0x40800000`
- Enable the capability to override MU-MIMO based on the vendor OUI.  
`iwpriv ath0 mucapwar 1`
- When no MU-capable stations are present, the dedicated client joins.  
The dedicated client joins as SU-2x2.

- When an MU-CAP normal client joins, the dedicated client must change as MU-1x1.  
The dedicated client shifts to be an MU-1x1.
- When an MU-CAP normal client leaves, the dedicated client must revert to 2x2.  
The dedicated client returns to be an SU-2x2.
- When the second dedicated-client joins, both the dedicated-clients become 1x1.  
Both dedicated clients become MU-1x1.
- When the second dedicated-client leaves, the remaining one becomes 2x2.  
The remaining dedicated client changes to SU-2x2.
- When a legacy client (without MU support) joins, the MU-capable phone must remain at 2x2.  
The dedicated client remains as 2x2
- When a deauthorization of all stations (including legacy clients) is performed, and a legacy client joins, the MU-capable client must join as 2x2.  
The dedicated client becomes SU-2x2.
- After the WAR feature is disabled and a phone that is in SU-2x2 mode when it is the only MU-capable client joins as the only client, enable the WAR feature.  
The dedicated client becomes SU-2x2.
- With the WAR feature enabled, make the MU-capable phone join as the only client (2x2), and then disable the WAR feature.  
The dedicated client becomes MU-1x1.

## 10.20 QCache

The purpose of QCache feature is to enable more peers in the system. Assuming that many of the peers are inactive most of the time, the firmware would swap out their state information into the host memory, freeing up the precious target memory to be used by more active peers. When an inactive peer starts to receive or send data, it transitions to an active state, and its state information is brought back from the host memory to the target memory.

The peers that match any one of the following criteria are never swapped out to the host memory and are always present in the target memory:

1. Self peer
2. Peer is being deleted
3. Peer is in disassociated state
4. BSS peer (AP)
5. VoW peer
6. Peer uses four addresses
7. Peer has data in the hardware queue

Peers that are in the target memory and do not match any of the above criteria could be swapped out to the host based on their inactivity time. In other words, peers that did not receive or transmit data for the longest period of time is evicted first.

The QCache feature is implemented entirely in the firmware, but must be statically enabled in the host driver. To enable the QCache feature, the host driver must be built with PEER\_CACHEING\_HOST\_ENABLE set to 1 (default option).

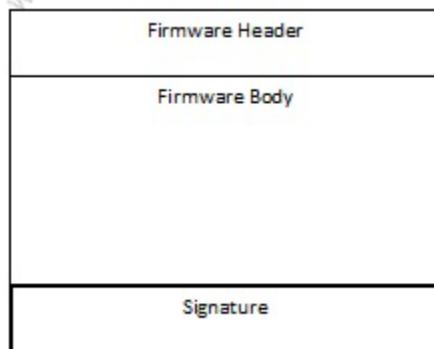
## 10.21 Firmware code sign and firmware authentication at host

Firmware authentication code can either exist along with host Wi-Fi software or can exist like another module that exports its functionality through set of APIs. At this point of time, this is planned to be implemented as independent module that provides all the functionality through APIs to the Wi-Fi software.

Firmware file authentication is two-step process. First step is to generate proper firmware headers and the signatures. Signatures of the firmware files would be generated through CASS tools. These would generate only digital signature of the files. Packaging of the generated signature along with the firmware file is needed to be done by this module.

Firmware file would be appended with F/W header, signature. Appending headers and trailers to the firmware image is done by a tool fw\_pack which is part of this software. This builds combined firmware binary with header and the signature at end.

Signed-firmware contains three parts, (1) the firmware header, (2) actual firmware body and the (3) signature. The following figure shows these components:



Crypto code from Linux kernel are used to authenticate the firmware signature. Next section would outline the header that would be added to the binary files.

### 10.21.1 Firmware file header format

All the firmware files that are resident on the flash or resident on file system must contain a header that properly identifies the different characteristics of the firmware file. Every file contains the details described in this subsection. The header contains all the relevant data to parse or signature.

When host build is generated, if feature is enabled by default, all firmware files would be encoded with respective headers. A packaging tool would be used would help in preparing a combined firmware file with header and signature. The signature itself is obtained in different methods as discussed here.

```

struct firmware_head {
    /* image magics and file identification */
    uint32          fw_img_magic_number;
    uint32          fw_chip_identification;
    uint32          fw_img_file_magic;
    uint16          fw_img_sign_ver;
    uint16          fw_img_spare1;
    uint32          fw_img_spare2;
    /* Versioning and release types */
    uint32          fw_ver_rel_type;
    uint32          fw_ver_maj;
    uint32          fw_ver_minor;
    /* image versioning is little tricky to handle. We assume there are three
     * different versions that we can encode.
     * MAJOR - 16 bits, lower 16 bits of this is spare, chip version encoded
     *           in lower 16 bits
     *           * minor - 16 bits, 0-65535,
     * sub-release - 16 bits, usually this would be zero. if required we
     * can use this. For eg. BL.2_1.400.2, is, QCA9980, 2.0, first major
     * release, build 400, and sub revision of 2 in the same build
     */
    uint32          fw_ver_spare3;
    /* header identificaiton */
    uint32          fw_hdr_length;
    uint32          fw_hdr_cksum;
    uint32          fw_hdr_flags;           // extra image flags
    // image flags are different single bit flags of different characterstics
    // At this point of time these flags include below, would extend as
    // required
    //                     signed/unsigned,
    //                     bin/text,
    //                     compressed/uncompressed,
    //                     unencrypted,
    //                     target_resident/host_resident,
    //                     executable/non-executable
    uint32          fw_hdr_spare4;
    uint32          fw_img_size;
    uint32          fw_img_length;
    /* there is no real requirement for keeping the size, the size is
     * fw_img_size = fw_img_length - ( header_size + signature)
     * in otherwords fw_img_size is actual image size that would be
     * downloaded to target board.
     */
    uint32          fw_spare5;
    uint32          fw_spare6;
    /* security details follows here after */
    uint8           fw_sig_algo; /* signature algorithm */
    uint16          fw_sig_len; /* index into known signature
lengths */
    uint8           fw_oem_id;      /* oem ID, to access otp or
so on */
}

```

```

    /* actual image body      */
    uint8          fw_img_body[fw_img_size];
    uint8          fw_img_padding[if_any_upto_4byte_boundary];
    uint8          fw_signature[256];
}

```

### **fw\_img\_magic\_number**

Assumption is every firmware can be identified with its firmware magic number. This bit OR of hex number of the product name. For QCA9980, general code word is 'BLNR', which is 0x424C4E52. Rest of the chips and their magic numbers can be found in code. General guidance is to define a 4 letter word that can fit in 16 bits of data.

### **fw\_chip\_identification**

PCI device ID for this chip. At this point this is defined to be only PCI device id excluding vendor id (168c). At same time this can be extended to contain the chip id too. At this point of time, the following chip identification numbers are defined.

- QCA9880 - 0x32
- QCA9980 - 0x40
- QCA9984 - 0x50
- QCA9886 - 0x56

### **fw\_img\_sign\_ver**

Backward compatibility might be required to support older signing method, as code signing feature evolves. This identifies the combined version of the tools, signing methods used at the time of signed firmware generation. At every major revision, code would be compiled with current software version. Depending on requirements, either files are authenticated or rejected.

### **fw\_ver\_rel\_type**

Each of the firmware can be either test firmware or production firmware. The keys/certificates used for authentication depends on the release status of the firmware files. All test firmwares would continue to use dev/test keys/certificates for signing and sign validation. All production/release firmware images should be using the production/release keys/certificates.

### **fw\_ver\_maj**

Typically, the format of the firmware release tags is 'CNSS.BL.3.0-00094-S-1'. At this point, 'CNSS' is not encoded anywhere in the header. For example, the above firmware is a retail release 1 (S-1), with major version 3, minor version 0, and build number 00094. Entire this information can be encoded in two 32 bit quantities. Note that first two members already identifying the product name and numbers. Product major release number is encoded in higher 16 bits of the fw\_ver\_maj and retail/enterprise version in lower 16 bits. The encoding almost looks like MMMM.MMMM.SSEE.1111 . SSEE actually would identify the build types, retail (S), Enterprise (E) and any other eg, carrier could use (C) which is not in use. These types are defined in code.

### **fw\_ver\_minor**

This parameter identifies the build number (00094 in the example presented earlier in this section). Both major and minor number are encoded in host software to enable rollback support. If there any host software release happens with particular combination, the firmware version is encoded into host software too. Also, the host software itself contains the host software information. This detail is for documentation purposes. No rollback support is available for host driver.

### **fw\_hdr\_length**

This parameter signifies the total length of the firmware header. Although this parameter is not essential, it is useful to compute checksum of the header to detect firmware corruptions early.

### **fw\_hdr\_cksum**

This parameter is the 1st complement addition of firmware header only, and is useful to detect header corruptions.

### **fw\_hdr\_flags**

Header flags add extra information on how to process this file. The header flags contain the following details:

- b'0 - 1 bit - signed(1) OR unsigned(0) - A bit field carries 1 if the file is signed and 0 if the file is unsigned. This bit is mandatory for the signature validation process.
- b'1-3 – 3 bits - binary or any other format, right now there are only two known formats, binary, and text
- b'4 - 1 bit - compressed OR uncompressed - Right now only uncompressed binary files are supported, if in future required, we can support compressed with proper compression algorithm information encoded either header or in other bits.
- b'5 - 1 bit - encrypted OR unencrypted - A firmware file can either be encrypted or not encrypted. At this point of time, this version only supports non-encrypted files, this can be extended to support encrypted f/w files.
- b'6 - 1 bit - is target downloaded (1) OR resident at host (0)
- b'7 1 bit - is executable (1) OR non-executable (0) - At this point this information is not useful for this module. This is for more documentation purpose.

### **fw\_img\_size**

This is actual number of firmware bytes without taking any padding into account. *fw\_img\_length*:  $fw\_img\_length = \text{sum}(fw\_img\_size, fw\_hdr\_length, padding, signature)$ , ideally this should match to the actual file size on disk with all the information.

### **fw\_sig\_algo**

This identifies the algorithm/security protocol that is used for encrypting the hash of the firmware file. At this point, SHA256 is used to compute the hash of the entire file including the header of the file. Encryption is done by RSS-PSS1 encryption method. The length of the hash is constant at 256

bytes (2048 bits) and attached at end of the file. Actual definitions of these codes would be updated in this design doc after code completion.

### **fw\_sig\_len**

Depending the hashing algorithm and the security cipher used to encrypt this length would change. At this point this is fixed at 2048 bits.

### **fw\_oem\_id**

The OEM identifier represents the keys to use from OTP. Because the host Wi-Fi software cannot access to OTP on board, it is assumed that this is not in use. The expectation from customer is to update the certificates/keys inside this software to appropriate keys before the firmware images are signed. If the customer requires to use OEM identifies from OTP and keys from OTPs/fuses, the customer must extend Linux crypto/TZ capabilities to access keys/OTP bits.

## **10.21.2 Firmware sign validation**

Host Wi-Fi software uses APIs (as mentioned in this document) to validate the firmware signature. The assumption is that firmware contains all respective headers. The following steps occur for firmware signature validation:

1. Check the header of the firmware file and complete sanity checks. Below sanity checks are done
  - a. Check if the firmware singing version is supported, if not empty the buffer and return error to the caller.
  - b. Check against DeviceID with encoded chipID in firmware file. Mostly this is redundant, yet it is good to check if end user copies a different file.
  - c. Check against known firmware magic numbers, known file types are
    - i. FW\_MAGIC\_BOARD\_DATA\_FILE
    - ii. FW\_MAGIC OTP FILE
    - iii. FW\_MAGIC\_WLAN\_BIN\_TARGET (athwlan.bin)
    - iv. FW\_MAGIC\_WLAN\_CODE\_SWAP (codeswap.athwlan.bin)
    - v. FW\_MAGIC\_WLAN\_OTHER\_TEST (This would be useful for other test firmware)
  - d. Check if the firmware is test or production, both host authentication software and the signed version should both be either production or test. If host authentication software is set to test mode, all firmware's are allowed
  - e. Check if firmware roll back feature is enabled, if enabled, read the combined version magic number with the one compiled in. If the version magic is lower than the one stored, fill the memory with zeros and return error. Following are the order of checks.
    - i. If (fw\_major >= stored\_fw\_major) && (fw\_minor >= stored\_fw\_minor) continue with rest of the process, otherwise return error with zeroing the memory.

2. Check firmware flags, if file is not singed, still had header, change the firmware start pointer pointing to the offset of the start of the firmware file, and return success
3. Check if file is compressed, if so return error ( scope for compressed firmware), if not proceed through next step
4. Check sanity on firmware file length and header length to make sure that they match. If not return error
5. Once all sanity passes through, call Linux Crypto API with all the necessary information.
6. If crypto API returns failure, fill the memory with zeros and return error to host Wi-Fi software.
7. Host Wi-Fi software should load the firmware to target only when it receives success (zero return value) and shall fail if error is returned. t

Figure 10-9 explains the entire flow.ckif y

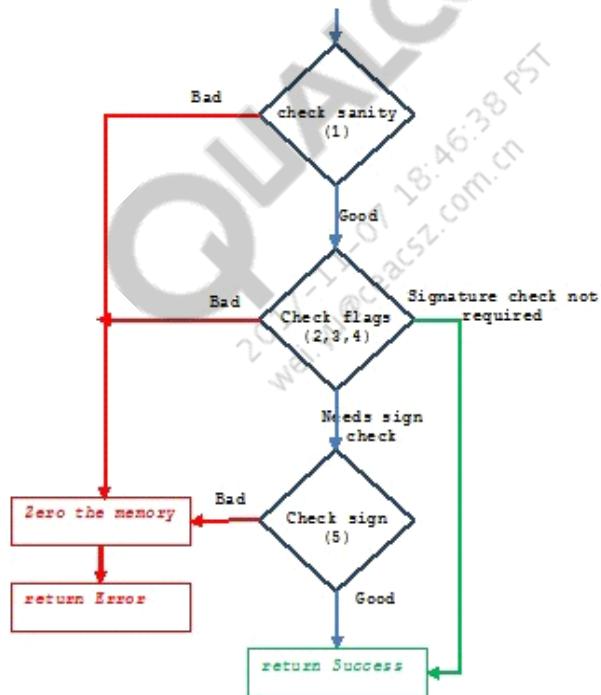


Figure 10-9 Firmware signature validation high level flowchart

### 10.21.3 API specification

UMAC module uses `request_firmware()` to load the firmware from linux kernel. A new API `request_secure_firmware()` on top of `request_firmware()` is added to keep the seamless integration with current code. This function would in turn uses `request_firmware()` and `authenticate_fw()` to check the signature of the file. On successful authentication, `request_secure_firmware` would return pointer to firmware memory, on failure, it returns NULL pointer.

## 10.22 WLAN LED implementation

### 10.22.1 Direct attach implementation for WLAN LED control

ATH\_SUPPORT\_LED is build flag to enable WLAN LED functionality for Direct attach cards. The older version of direct attach pci cards used ath\_led\_report\_data\_flow and ath\_led\_heartbeat function defined in ath\_led.c for software based LED control. Different customer uses different blinking rate table which can be configured in wlan build config file.

Integrated direct attach cards like DB120, AP135, AP147, AP152 have hardware based LED control. Just configuring the ATH\_GPIO\_OE and relevant ATH\_GPIO\_OUT\_FUNCTION ensure hardware control of LED.

### 10.22.2 Offload path implantation for WLAN LED control

OL\_ATH\_SUPPORT\_LED is the build flag to enable WLAN LED functionality. There are two timers used for WLAN LED functionality.

```
static OS_TIMER_FUNC(ol_ath_led_blink_timed_out);
static OS_TIMER_FUNC(ol_ath_led_poll_timed_out);
```

ol\_ath\_led\_poll\_timed\_out blinks at constant rate ol\_ath\_led\_blink(scn, 100, 500). Poll timer is called even when there is no Tx/Rx traffic. ol\_ath\_led\_blink\_timed\_out is called when there is Tx/Rx traffic. Based on scn > scn\_led\_byte\_cnt the blink rates are programmed. And once the blink rates are programmed scn\_led\_byte\_cnt is set to 0.

For Tx path ol\_tx\_completion\_handler and ol\_tx\_inspect\_handler update the scn > scn\_led\_byte\_cnt and call the ol\_ath\_led\_event.

For Rx path ol\_rx\_deliver and ol\_rx\_deliver\_raw update the scn > scn\_led\_byte\_cnt and call the ol\_ath\_led\_event which in turn calls ol\_ath\_led\_blink\_timed\_out timer for blinking.

Blinking is done through setting 0 and 1 for wmi\_unified\_gpio\_output so FW toggles the WLAN LED GPIO.

Each rate is 50 Mbps per entry in increasing order.

```
static const OL_LED_BLINK_RATES ol_led_blink_rate_table[] = {
    { 500, 130 },
    { 400, 100 },
    { 280, 70 },
    { 240, 60 },
    { 200, 50 },
    { 160, 40 },
    { 130, 30 },
    { 100, 30 },
    { 90, 20 },
    { 80, 20 },
    { 70, 20 },
    { 60, 10 },
    { 50, 10 },
    { 40, 10 },
};
```

GPIO PIN for WLAN LED is configured in scn\_led\_gpio based on target\_version / target\_type in ol\_ath\_attach.

```
#define PEREGRINE_LED_GPIO      1
#define BEELINER_LED_GPIO       17
#define CASCADE_LED_GPIO       17
#define BESRA_LED_GPIO        17
#define IPQ4019_LED_GPIO       5
```

IPQ40xx which is integrated Wi-Fi does not use wmi\_unified\_gpio\_output but uses ipq4019\_wifi\_led to set the GPIO directly from the host. Other functionality remains same as other chips.

## 10.23 Pre-allocation of the required runtime memory

All the runtime dynamic allocations can be changed to allocate from pre-allocated pools. These pools are based on type of data-structures. This approach eliminates any additional allocations at run-time from available system memory, once the AP is up and running.

Currently, the driver preallocates memory for a specified number of VAPs, associating clients and scan entries. The default values are 16 VAPs, 124 clients and 256 scan entries.

This pre-allocation is disabled by default and to enable it, prealloc\_disabled=0 must be passed as module param. The module param, max\_vaps can be used to specify the maximum number of VAPs that needs to be pre-allocated.

### 10.23.1 Memory pool data structures

The following data structures are added to cmn\_dev/qdf/linux/src/i\_qdf\_mem.h, to manage memory pools:

```
#define MAX_MEM_POOLS 256
typedef struct mempool_elem {
    STAILQ_ENTRY(mempool_elem) mempool_entry;
} mempool_elem_t;
typedef struct __qdf_mempool_ctxt {
    int pool_id;
    u_int32_t flags;
    size_t elem_size;
    void *pool_mem;
    u_int32_t mem_size;
    STAILQ_HEAD(, mempool_elem) free_list;
    spinlock_t lock;
#ifndef MEMPOOL_DEBUG
    u_int32_t max_elem;
    u_int32_t free_cnt;
#endif
} qdf_mempool_ctxt_t;
typedef mempool_ctxt_t * __qdf_mempool_t;
```

The following change has been added to cmn\_dev/qdf/linux/src/i\_qdf\_types.h:

```
Struct __qdf_device {
    ...
```

```
struct __qdf_mempool_ctxt *mem_pool[MAX_MEM_POOLS];
#endif
};
```

### 10.23.2 Initialization of memory pools

The following interfaces are defined to initialize various memory pools required for run-time usage. Pool initialization interfaces are added to the existing cmn\_dev/qdf/inc/qdf\_mem.h

```
int qdf_mempool_init(qdf_device_t osdev, qdf_mempool_t *pool_addr, int
elem_cnt, size_t elem_size, u_int32_t flags);
void qdf_mempool_destroy(qdf_device_t osdev, qdf_mempool_t pool);
```

Following definitions are added to Linux specific file cmn\_dev/qdf/linux/src/qdf\_mem.c

```
/**
 * @brief Create and initialize memory pool
 *
 * @param[in] osdev platform device object
 * @param[out] pool_addr address of the pool created
 * @param[in] elem_cnt no. of elements in pool
 * @param[in] pool_elem_size size of each pool element in bytes
 * @param[in] flags flags
 *
 * @return Handle to memory pool or NULL if allocation failed
 */
int __qdf_mempool_init(qdf_device_t osdev, qdf_mempool_t *pool_addr,
int elem_cnt, size_t pool_elem_size, u_int32_t flags)
{
    /* - Allocate memory for pool context
     * - Round up the element size to cache-line size.
     * - Allocate pool memory and align start address to cache-line
     * boundary
     * - Initialize pool lock and free list */
}
/***
 * @brief Destroy memory pool
 *
 * @param[in] osdev platform device object
 * @param[in] Handle to memory pool
 */
void __qdf_mempool_destroy(qdf_device_t osdev, __qdf_mempool_t pool)
#if PREALLOC_DEBUG
    /* Check if free count matches pool_cnt if debug is enabled */
#endif
    /* Free pool memory and pool context memory */
}
#endif
```

### 10.23.3 Runtime allocation and free interfaces

Following interfaces are defined to allocate and free run-time memory from pre-allocated pools. Pool allocation interfaces are added to the existing file cmn\_dev/qdf/inc/qdf\_mem.h

```
/***
```

```

* @brief Allocate an element memory pool
*
* @param[in] osdev  platform device object
* @param[in] Handle to memory pool
*
* @return  Pointer to the allocated element or NULL if the pool is empty
*/
void *qdf_mempool_alloc(qdf_device_t osdev, qdf_mempool_t pool);

/**
* @brief Free a memory pool element
*
* @param[in] osdev Platform device object
* @param[in] pool  Handle to memory pool
* @param[in] buf   Element to be freed
*/
void qdf_mempool_free(qdf_device_t osdev, qdf_mempool_t pool, void *buf);

```

If the feature is disabled, these are mapped to regular allocations `qdf_mem_alloc` and `qdf_mem_free`.

## 10.24 Memory Footprint Reduction on QCA9880/QCA9886/QCA9889

This feature adds support for entry level platform (*Low memory AP151/AP147 + QCA9880/QCA9886/QCA9889 platform with 8M Flash + 64 M DDR support*) and is responsible for memory footprint reduction, so the entry level platform can perform WiFi operations in 64M DRR without any failures and can accommodate all necessary packages to be installed in 8M flash memory.

Because the entry level platform supports only 8M flash with 64M DDR, below actions are followed for both DDR and Flash memory footprint reduction.

- Limited the number of VAP and client support per Radio.
- Disabled packet log feature by default for entry level platforms. Users can enable it by setting enable option to 1 in wireless configure file and reloading the wifi modules.
- Disabled some WiFi Host Driver and OpenWrt features to achieve flash memory usage reduction.
- Enabled some of the OpenWrt packages as loadable packages for entry level platform Flash memory footprint reduction. These modules will not be installed by default. Users have to install the packages manually, as needed, after the image is being flashed as per the need.
- Target FW binaries and utility packages are being introduced as loadable packages for entry level platform.
- Like in other AP151/AP147 + QCA9880/QCA9886/QCA9889 platforms, the firmware memory will not be saved for debugging on firmware target assert. Crash dump support is also disabled.

The following is the general overview of the 8M flash memory layout for *Low memory AP151/AP147 + QCA9880/QCA9886/QCA9889 platform*:



Steps for testing enabling debug build (Crash scope enabled) Images in entry level platform:

- Use a 16 MB flash + 128 MB platform and configure it with 8 M Flash and 76 M DDR support. This cannot be done in a 64 MB platform because the free memory after reserving for crash scope is not adequate to some of the tests after Wi-Fi up.
- Then the Debug build Image for entry level platform has to be flashed on the 16 MB flash + 128 MB DDR board, using 8MB+64MB DDR flash instruction.

### Wi-Fi Host changes for DDR memory footprint reduction

- Added new Wireless profile ‘`config.wlan.lowmem.profile`’ which will be used for entry level platform instead of **unified perf/profile**.
- Support for number of AP VAP and Clients has been reduced to 2 AP VAP and 16 Clients. By doing this, in case of Offload chipsets, *we will be able to save more than 1MB memory*.
- TX, QOS NULL and RX descriptors have been reduced in case of direct attach chipsets for run time memory optimization, *which can save up to 2.5MB memory*.
- Reduced HTT RX Ring buffer size from 2048 to 1024 and Ring fill up level from 1000 to 500 for Offload Radio, *which can save up to 2MB memory*.
- Added a new module parameter to enable or disable packet log during WiFi load time instead of disabling packet log feature at compile time. The advantage towards memory optimization in case of Low memory platform is by disabling packet log, HOST Driver will not allocate and push any SKB buffer for Copy Engine 8 destination ring, *which in return save around 512 KB*.
- If `WLAN_FEATURE_FASTPATH` is enabled for entry level platform, to reduce the run time memory, copy engine 1 destination ring receive buffer entries has been reduced from 512 to 64. If `WLAN_FEATURE_FASTPATH` is enabled, the target to host HTT message processing is handled by copy engine 5 instead of copy engine 1. *Which in return save around 448 KB*.
- For Offload Chipsets the number of packets queued before giving to FW has been reduced to 4K from 8K for entry level platform, i.e during heavy traffic run the platform can hold up to 4096 transmitted buffers without dropping the packet. Any more packets will be dropped.
- The following is the list of preprocessor Macros and Variable in Host Driver that are either got reduced or manipulated in order to get above functionality.

|                           |   | Original Value         | Changed Value        |
|---------------------------|---|------------------------|----------------------|
| No of VAP Support         | ATH_BCBUF<br>CFG_TGT_NUM_VDEV_QCA9888<br>CFG_TGT_NUM_VDEV_QCA9984<br>CFG_TGT_NUM_VDEV_AR900B<br>CFG_TGT_NUM_VDEV_AR988X | 17<br>17               | 4<br>4               |
| No of Clients Support     | ic_num_clients<br>CFG_TGT_QCACHE_ACTIVE_PEERS<br>CFG_TGT_QCACHE_MIN_ACTIVE_PEERS<br>CFG_TGT_NUM_QCACHE_PEERS_MAX        | 127<br>35<br>26<br>513 | 33<br>20<br>16<br>33 |
| DA QOS NULL TX Descriptor | ATH_QOSNULL_TXDESC  | 64                     | 32                   |
| DA RX Descriptor          | ATH_RXBUFF  | 1024                   | 512                  |
| HTT RX Ring Size          | OL_CFG_RX_RING_SIZE_MAX   | 2048                   | 1024                 |
| HTT RX Ring Fill up Level | htt_rx_ring_fill_level  | 1024                   | 512                  |
| Copy Engine 1 Descriptor  | CE 1 Descriptor   | 512                    | 64                   |
| TX Packet Queue           | OL_TX_FLOW_CTRL_QUEUE_LEN<br>OL_TX_PFLOW_CTRL_MAX_BUF_GLOBAL  | 8K                     | 4K                   |

### OpenWrt/ Linux changes for DDR memory footprint reduction

- SKB\_RECYCLER disabled in OpenWrt configuration for entry level platform.  
SKB allocation logic is changed to allocate the actual requested length during a skb allocation call, instead of allocating a standard size of 4k. This helps to reduce the overall memory consumption, especially if the actual size requested is less than 2K.
- Support to use RAM size from u-boot has been added for entry level platform, in which case Kernel will only utilize the RAM size which is passed from u-boot. This is helpful for testing the entry level platform Debug build (Crash scope enabled) Images in 16MB flash + 128 MB platform, which will be configured and loaded with 8M Flash and 76MDDR support.

### WiFi Host changes for Flash memory footprint reduction

- For flash memory footprint reduction some of the WiFi features are disabled during compile time for entry level platform. Removal has been done because most of the following features not only increase the Host Driver package size, but also need user space utility to provide the support.

```
AH_SUPPORT_AR5416=0
ATH_SUPPORT_SPECTRAL=0
ATH_SUPPORT_WIFIPOS=0
UMAC_SUPPORT_WIFIPOS_LEANCC=0
ATH_SUPPORT_HYFI_ENHANCEMENTS=0
ATH_SUPPORT_WRAP=0
UNIFIED_SMARTANTENNA=0
```

```
UMAC_SUPPORT_ACFG=0
QCA_AIRTIME_FAIRNESS=0

ATH_BAND_STEERING=0
ATH_TX_OVERFLOW_IND=0
ACFG_NETLINK_TX=0
ACFG_MON_FILTER=0
WMI_RECORDING=0
REMOVE_INIT_DEBUG_CODE=1
ATH_RXBUF=256
ATH_SUPPORT_LOWI=0
WMI_INTERFACE_EVENT_LOGGING=0

open_ssl_utils=n
mtd_utils=n
sigmadut=n
qca_wrapd=n
qca_wpc=n
qca_spectral=n
qca_acfg=n
luci=n
luci_whc=n
hyfi=n
kmod-fs-ext4=n
kmod-fs-msdos=n
kmod-fs-vfat=n

qca_wsplcd=n
rp_ppoe_relay
qcagga=n
mcproxy=n
ddns_scripts=n
miniupnpd=n
lua=n
kmod_fast_classifier=n
radvd=n
whc=n
qca-wsplcd=n
qca-ieee1905-init=n
kmod-ipt-nathelper-extra=n

qca-ieee1905-gn=n
qca-hyd=n
tc=n
rp-pppoe-relay=n
quagga=n
miniupnpd=n
iptables-mod-mark2prio=n
iptables-mod-ipopt=n
iptables-mod-conntrack-extra=n
curl=n
qca-vhyfid=n
qca-hyd-plc=n
```

## OpenWrt changes for Flash memory footprint reduction

- For flash memory footprint reduction some of the OpenWrt features are being disabled during compile time
- Below are the list of OpenWrt packages/features that are being removed from OpenWrt configuration for entry level platform to free up flash memory.
- Some other OpenWrt packages are enable as loadable packages, to accommodate all necessary packages to be installed in 8M flash memory
- The following is the list of OpenWrt packages that are enabled as loadable packages for entry level platform to free up flash memory. Engineer has to install the packages manually after the image is being flashed as per the need.

```
qca-ssdk_shell=m          ntfs_3G=m
qcmbr=m                  kmod_art2=m
samba36_server=m         qca-wapid=m
qca-wifi-fw-hw9-10.4-asic=m qca-wifi-fw-hw7-10.4-asic=m
qca-wifi-fw-hw6-10.4-asic=m qca-wifi-fw-hw3-10.4-asic=m
qca-wifi-fw-hw10-10.4-asic=m
```

## Free Memory detail with and without memory footprint reduction

- Without WIFI host changes for Flash memory foot print reduction there will be 690K flash memory available after flashing the Image with pre-installed target binaries.
- The following is the Free DDR (RAM) memory detail without DDR memory foot print reduction. Here the skb recycler is enabled in compilation but the number of skbs is configured to a very low value of 1.

|                       | <b>MemTotal<br/>(in kB)</b> | <b>MemFree<br/>(in kB)</b> | <b>Buffers<br/>(in kB)</b> | <b>Cached<br/>(in kB)</b> | <b>SKB Recycler<br/>Buffer count</b> |
|-----------------------|-----------------------------|----------------------------|----------------------------|---------------------------|--------------------------------------|
| Wifi Load With no VAP | 61364                       | 6856                       | 4004                       | 14160                     | 1                                    |
| 1 2 GHz + 1 5 GHz VAP | 61364                       | 2044                       | 3916                       | 14020                     | 1                                    |
| 2 2 GHz + 2 5 GHz VAP | 61364                       | 2660                       | 3964                       | 14416                     | 1                                    |

- After making Wi-Fi host changes for flash memory foot print reduction there will be 888K flash memory available after flashing the Image and manually installing the target binaries.
- The following is the Free DDR (RAM) memory detail with DDR memory foot print reduction.

|                       | <b>MemTotal<br/>(in kB)</b> | <b>MemFree<br/>(in kB)</b> | <b>Buffers<br/>(in kB)</b> | <b>Cached<br/>(in kB)</b> |
|-----------------------|-----------------------------|----------------------------|----------------------------|---------------------------|
| Wifi Load With no VAP | 61364                       | 19024                      | 3792                       | 13484                     |
| 1 2 GHz + 1 5 GHz VAP | 61364                       | 18532                      | 3792                       | 13648                     |
| 2 2 GHz + 2 5 GHz VAP | 61364                       | 18204                      | 3792                       | 13656                     |

## 10.25 RDK-B HAL API PHASE 2

Following are the API implemented:

```
wifi_getApAssociatedDevicesHighWatermarkThreshold
wifi_setApAssociatedDevicesHighWatermarkThreshold
wifi_getApAssociatedDevicesHighWatermarkThresholdReached
wifi_getApAssociatedDevicesHighWatermark
wifi_getApAssociatedDevicesHighWatermarkDate
wifi_setRadioTrafficStatsMeasure
wifi_getRadioStatsReceivedSignalLevel
```

### 10.25.1 wifi\_getApAssociatedDevicesHighWatermarkThreshold

This will get the high watermark threshold value which is set by the user or the default value. The HighWatermarkThreshold value that is lesser than or equal to MaxAssociatedDevices. This holds a value of MaxAssociatedDevices or 50. The default value of this parameter should be equal to MaxAssociatedDevices. In case MaxAssociatedDevices is 0 (zero), the default value of this parameter should be 50. A value of 0 means that there is no specific limit and Watermark calculation algorithm should be turned off.

### 10.25.2 wifi\_setApAssociatedDevicesHighWatermarkThreshold

This will set the threshold as given by the user taking into account of the boundary conditions mentioned above.

### 10.25.3 wifi\_getApAssociatedDevicesHighWatermarkThresholdReached

The HighWatermarkThresholdReached will denote the number of times the maximum threshold is reached. Say for example the high watermark threshold is 40 and Max associated devices is 50 and the number of devices connected is 40 and then it goes down to 25 and then again to 40. So now the HighWatermarkThresholdReached will return 2, since it has breached the threshold twice.

### 10.25.4 wifi\_getApAssociatedDevicesHighWatermark

For getApAssociatedDevicesHighWatermark the maximum number of associated devices that has associated since the last reset. For example if the high watermark threshold is 40 and the maxassociated devices is 50 and the maximum number of devices connected is 45. Then this API should return 45.

### 10.25.5 wifi\_getApAssociatedDevicesHighWatermarkDate

For getApAssociatedDevicesHighWatermarkDate the time at which the maximum number of associated devices ever associated since the last reset. The system's time will be given.

## 10.25.6 wifi\_setRadioTrafficStatsMeasure

The wifi\_setRadioTrafficStatsMeasure allows the user to set the measuring rate and measuring interval. The measuring rate denotes the time at which the RSSI is to be measured and measuring interval denotes the interval for which the stats has to be measured and displayed to the user at the end of this interval.

## 10.25.7 wifi\_getRadioStatsReceivedSignalLevel

This API requires the measuring rate and measuring interval parameters and the signal level is displayed at the end of measuring interval period

Assume the measuring rate is 300s and measuring interval is 1800 then the number of bins is 6 i.e at every 300s the RSSI is measured and the bin is filled and the min, max and median is calculated at every 300s and displayed at the end of 1800s.

A sample of how the bins will look is mentioned in the following table:

| Time | Noise in bin      | Maximum | Minimum | Median |
|------|-------------------|---------|---------|--------|
| 300  | 0,0,0,0,30        | 30      | 30      | 30     |
| 600  | 0,0,0,30,33       | 33      | 30      | 33     |
| 900  | 0,0,0,30,33,25    | 33      | 25      | 25     |
| 1200 | 0,0,30,33,25,14   | 33      | 14      | 25     |
| 1500 | 0,30,33,25,14,27  | 33      | 14      | 27     |
| 1800 | 30,33,25,14,27,18 | 33      | 14      | 27     |

This data is maintained on per node basis.

## 10.26 Wi-Fi reload

### 10.26.1 Avoiding module reload during wifi up/down

Instead of reloading all the wifi modules during wifi up/down, reloading a firmware helps in reducing memory free/alloc cycles. During the last VAP removal, the firmware is unloaded and during the first VAP creation, the firmware is re-initialized.

Reloading a firmware is targeted to achieve a better utilization of the allocated memory and avoid fragmentation due to frequent free/alloc cycles. This design is also meaningful since, 'wifi' command is mainly used to effect the changed configuration. If that can be achieved without reloading modules, then, it would really be a better design.

## 10.26.2 Code changes

Because the modules are not unloaded, both the firmware and host variables should be reset to default init values - as same as reloading modules.

### 10.26.2.1 Firmware reset

A new method/WMI command to reinitialize firmware is not implemented; instead, the approach is to reload the firmware. Firmware unload is performed during the last VAP removal and reloaded during the first VAP creation.

Code rearrangement is performed in the init path (pci, ahb probe, configure and ol\_ath\_attach) and remove path (ol\_ath\_detach) by splitting into several smaller functions. These are called during firmware unload and reinitialization of the firmware.

### 10.26.2.2 Host variables reset

New functions are added to reset the ic, scn, sc structure variables to default values for both offload and direct attach architectures.

This reinit is performed during the firmware reload in OL path.

If this step is not performed, any uci/iwpriv param set during the previous test run is used as such in the current run (even after giving wifi) without initializing to default values.

## 10.26.3 Reloading the modules with new module parameters

'wifi load' is supported now, which reloads the modules for any new module parameter change. List of Module params for which 'wifi load' needs to be passed so that the modules are reloaded:

```
'testmode'
'veow_config'
'ol_bk_min_free'
'ol_be_min_free'
'ol_vi_min_free'
'ol_vo_min_free'
'ar900b_emu'
'frac'
'intval'
'atf_mode'
'atf_msdu_desc'
'atf_peers'
'atf_max_vdevs'
'fw_dump_options'
'enableuartprint'
'ar900b_20_targ_clk'
'qca9888_20_targ_clk'
'max_descs'
'max_peers'
'qwrap_enable'
'otp_mod_param'
'max_active_peers'
```

```
'enable_smart_antenna'
'nss_wifi_olcfg'
'max_clients'
'max_vaps'
'enable_smart_antenna_da'
'prealloc_disabled'
'lteu_support'
'enable_mesh_support'
'enable_mesh_peer_cap_update'
```

For other uci changes, normal wifi command is sufficient.

## 10.27 FILS capability

Support has been introduced for compatibility with the IEEE 802.11ai protocol to enable the implementation of fast initial link setup (FILS) capability on the proprietary Wi-Fi driver and the hostapd application.

FILS protocol defined in 802.11ai specification is helpful to reduce the connection delay. WLAN initial connection delay is reduced without any compromise on security. FILS uses EAP-RP protocol for authentication and key derivation. Association to a FILS-capable AP always occurs through a full EAP exchange. The subsequent connections use the FILS authentication algorithm to minimize the delay in connection.

### 10.27.1 Wi-Fi driver enhancements for FILS support

To enable FILS capability, enter the `iwpriv athX enable_fils 1` command. To disable FILS capability, enter the `iwpriv athX enable_fils 0` command. To retrieve the configured value, enter the `iwpriv athX g_enable_fils` command.

This option or parameter is added in the driver to provide backward compatibility support. In certain scenarios, the old driver (without FILS capability) is used with new hostapd (with FILS support). When the `enable_fils` parameter is entered, hostapd queries the driver for FILS support and stops performing FILS specific operations such as `IEEE80211_MLME_AUTH_FILS` for drivers that do not support FILS.

From the configuration perspective, a user must set a single UCI config option to turn on the FILS support in both driver and hostapd applications. To enable FILS capability in both driver and hostapd, enter the `uci set wireless.@wifi-iface[1].ieee80211ai=1` command. To disable FILS capability in both driver and hostapd, enter the `uci set wireless.@wifi-iface[1].ieee80211ai=0` command.

When FILS is compiled and when the feature is enabled, the EXT CAP field indicates FILS capability as defined in IEEE 802.11ai draft specification.

For FILS, the (Re)association frames are encrypted and handled separately in the driver. With the Wi-Fi driver, the association packet to be transmitted is formed in the driver; therefore, the encryption and decryption operations are performed in the driver. FILS uses Authenticated Encryption with Associated Data (AEAD) cipher mode to protect association frames. A new crypto algorithm is registered for this purpose. This crypto algorithm uses the AEAD parameters

as passed by the hostapd and follows software encryption and decryption. These encryption and decryption processes are applicable only for (Re)Association Request and Response frames.

The hostapd sends the AEAD parameters to the driver while sending Authentication Response. Note that the operation field set by the hostapd while sending Authentication response is different than other cases. This has been performed to identify Authentication frame holding AEAD parameters from regular ones. After receiving AUTH\_FILS operation, the driver sets the key required for the encryption algorithm. After receiving the association request with FILS, the FILS AEAD algorithm starts and decryption is carried out using the keys passed by the hostapd in the previous step. Similarly, the association response packets are also encrypted using the same parameters.

## 10.27.2 Hostapd enhancements for FILS support

The hostapd application determines whether the driver support FILS. If the driver does not support FILS, all operations related to FILS are ignored.

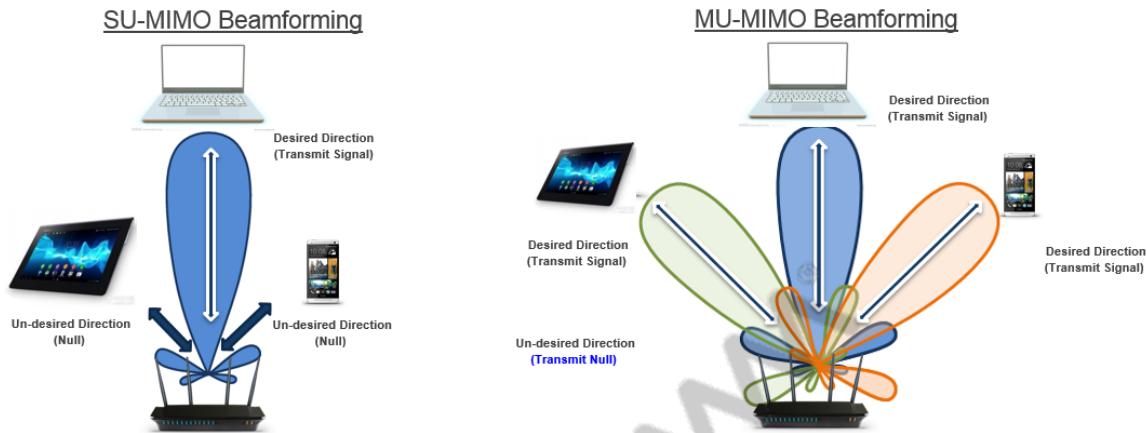
According to the 802.11ai specification, the FILS indication element is added to beacon and probe responses when 802.11ai support is enabled in hostapd

For authentication and association frame processing, the following enhancements are made:

- Send encryption parameters to the driver while sending Authentication response frames
- Insert FILS-specific information in the Authentication Response packet.
- Process the decrypted Association Response frame as sent by the driver to validate FILS fields
- Insert FILS-specific information in the Association Response packet.
- Set appropriate keys to the driver/hardware on successful authentication and association.

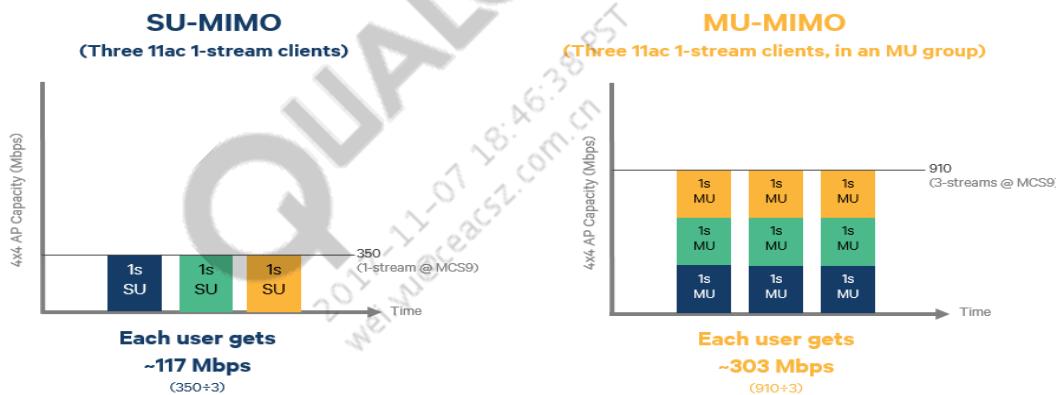
## 10.28 MU-MIMO

Multiple-input and Multiple-output (MIMO) is a system that uses multiple antennas at both the transmitter and receiver to improve communication performance. SU-MIMO involves transmitting one or multiple streams to one user at a time. MU-MIMO involves transmitting one or many streams to more than one user at a time in the same set of frequency resources

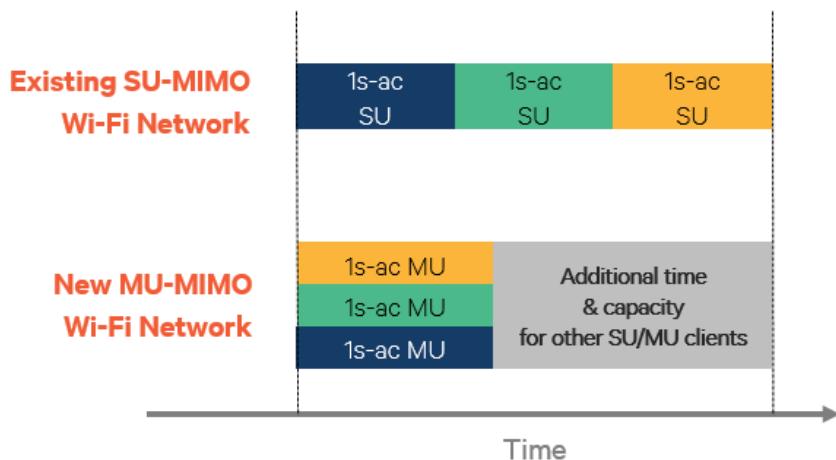


The following are the benefits of MU-MIMO over SU-MIMO:

- Supporting multiple clients at the same time can improve overall DL throughput per client.



- With 2 to 3x greater efficiency for MU clients, the network has more “free” time or capacity to serve other clients. Important for enterprise network.



## 10.28.1 MU-MIMO per VAP statistics

In the existing system, firmware supports radio based firmware Tx/Rx statistics and do not support VAP (Virtual AP) based firmware Tx/Rx statistics. The VAP based firmware Tx/Rx statistics feature is to have VAP based statistics tailored for Tx MU stats, Tx sounding stats, Tx Selfgen stats, in order to know Tx MU packets transmitted to a particular peer and to display user positions of a particular peer in different MU groups. Memory will be accommodated to maintain these stats in firmware only if the maximum number of peers is restricted to 240.

### 10.28.1.1 Known limitations and exceptions

Memory will be accommodated in firmware for these VAP based statistics by reducing maximum number of peers to 240 as agreed by customer. The control to reduce the number of peers to 240 is made available to the customer through UCI.

Execute the following command at the AP console

```
uci set wireless.qcawifi=qcawifi
uci set wireless.qcawifi.max_peers=240
uci commit
wifi
```

## 10.28.2 Tx sounding statistics

Execute the following command at the AP console.

```
iwpriv athx vap_txrx_stats 14
```

Give VAP ID within valid range e.g.: ath010, and if the VAP is created then it prints the stats for that particular VAP.

```
cbf_20[4];
cbf_40[4];
cbf_80[4];
cbf_160[4];
sounding[NUM_OF_SOUNDING_STATS_WORDS];
```

### Reset statistics

Execute the following command at the AP console.

```
iwpriv athx vap_txrx_st_rst 14
```

Give VAP ID within valid range. For example, ath010, and if the VAP is created then it resets the stats for that particular VAP.

## 10.28.3 Tx MU statistics

### Publish statistics

Execute the following command at the AP console.

```
iwpriv athx vap_txrx_stats 17
```

Give VAP ID within valid range e.g.: ath010, and if the VAP is created then it prints the following fields/stats for that particular VAP.

```
mu_sch_nusers_2;
mu_sch_nusers_3;
mu_mpdu_queued_usr[4];
mu_mpdu_failed_usr[4];
```

### **Reset statistics**

Execute the following command at the AP console.

```
iwpriv athx vap_txrx_st_rst 17
```

Give VAP ID within valid range e.g.: ath010, and if the VAP is created then it resets the stats for that particular VAP.

## **10.28.4 Tx Selfgen statistics**

Execute the following command at the AP console.

```
iwpriv athx vap_txrx_stats 16
```

Give VAP ID within valid range e.g.: at010, and if the VAP is created then it prints the following mentioned fields/stats values for that particular VAP.

```
su_ndpa;           /* num SU-NDPA frames transmitted */
su_ndp;            /* num SU-NDP frames transmitted */
mu_ndpa;           /* num MU-NDPA frames transmitted */
mu_ndp;            /* num MU-NDP frames transmitted */
mu_brpoll_1;       /* num BRPOLL frames transmitted to second user
*/
mu_brpoll_2;       /* num BRPOLL frames transmitted to third user
*/
su_bar;             /* num BAR frames transmitted in SU seq to flush
BA state */
su_cts;              /* num CTS2SELf frames to extend SU-data burst
*/
su_ndpa_err;         /* num SU-NDPA frames not transmitted due to HW
pause */
su_ndp_err;          /* num SU-NDP frames that didn't receive correct
CBF */
```

### **Reset statistics**

Execute the following command at the AP console.

```
iwpriv athx vap_txrx_st_rst 16
```

Give VAP ID within valid range e.g.: ath010, and if the VAP is created then it resets the stats for that particular VAP.

### 10.28.5 Tx transmitted MU packet count per peer

#### **Publish statistics**

Execute the following command at the AP console.

```
iwpriv ath0 get_mu_tx_count AID
```

Print Tx MU packets transmitted count for that peer.

#### **Reset statistics**

Execute the following command at the AP console.

```
iwpriv ath0 rst_mu_tx_count AID
```

Reset Tx MU packets transmitted count for that peer and later if you print the count it displays zero.

### 10.28.6 User positions in different MU groups for a particular AID

#### **Publish user position values**

Execute the following command at the AP console.

```
iwpriv ath0 get_mu_peer_pos AID
```

Print user positions of that particular peer in different MU groups with groupid 0 to groupid 15.

### 10.28.7 Sample test scenario

The following UCI configuration file can be referred to configure in maximum VAP mode to test per VAP statistics.

#### 10.28.7.1 Setting up AP

```
uci set wireless.wifi0=wifi-device
uci set wireless.wifi0.type=qcawifi
uci set wireless.wifi0.macaddr=8C:FD:02:00:c9:c9
uci set wireless.wifi0.hwmode=11ac
uci set wireless.wifi0.disabled=0
uci set wireless.wifi0.htmode=HT80
uci set wireless.wifi0.channel=100
uci set wireless.wifi0.txchainmask=15
uci set wireless.wifi0.rxchainmask=15
uci set wireless.wifi0.mode=ap
uci set wireless.@wifi-iface[0]=wifi-iface
uci set wireless.@wifi-iface[0].device=wifi0
uci set wireless.@wifi-iface[0].network=lan
```

```
uci set wireless.@wifi-iface[0].mode=ap
uci set wireless.@wifi-iface[0].ssid=5g_wrt
uci commit wireless
```

### 10.28.7.2 Setting up STA1

```
uci set wireless.wifi0=wifi-device
uci set wireless.wifi0.type=qcawifi
uci set wireless.wifi0.macaddr=84:60:aa:00:c9:cc
uci set wireless.wifi0.hwmode=11ac
uci set wireless.wifi0.disabled=0
uci set wireless.wifi0.htmode=HT80
uci set wireless.wifi0.channel=100
uci set wireless.wifi0.txchainmask=15
uci set wireless.wifi0.rxchainmask=15
uci set wireless.wifi0.mode=sta
uci set wireless.@wifi-iface[0]=wifi-iface
uci set wireless.@wifi-iface[0].device=wifi0
uci set wireless.@wifi-iface[0].network=lan
uci set wireless.@wifi-iface[0].mode=sta
uci set wireless.@wifi-iface[0].wds=0
uci set wireless.@wifi-iface[0].ssid=5g_wrt
uci commit wireless
```

### 10.28.7.3 Setting up STA2

```
uci set wireless.wifi0=wifi-device
uci set wireless.wifi0.type=qcawifi
uci set wireless.wifi0.macaddr=84:60:aa:00:c9:d3
uci set wireless.wifi0.hwmode=11ac
uci set wireless.wifi0.disabled=0
uci set wireless.wifi0.htmode=HT80
uci set wireless.wifi0.channel=100
uci set wireless.wifi0.txchainmask=15
uci set wireless.wifi0.rxchainmask=15
uci set wireless.wifi0.mode=sta
uci set wireless.@wifi-iface[0]=wifi-iface
uci set wireless.@wifi-iface[0].device=wifi0
uci set wireless.@wifi-iface[0].network=lan
uci set wireless.@wifi-iface[0].mode=sta
uci set wireless.@wifi-iface[0].wds=0
uci set wireless.@wifi-iface[0].ssid=5g_wrt
uci commit wireless
```

### 10.28.7.4 Configuring IP Address

After AP and STA associated, do following on STA1:

```
ifconfig br-lan 0.0.0.0
ifconfig ath0 192.168.1.10
```

After AP and STA associated, do following on STA2:

```
ifconfig br-lan 0.0.0.0
```

```
ifconfig ath0 192.168.1.11
```

### 10.28.7.5 Generating UDP traffic – AP → STA1 and AP → STA2

- Execute from STA1: *iperf -u -s -i 1*
- Execute from AP:
- *iperf -c 192.168.1.10 -u -b 1000M -i 1 -t 10000*
- 818 Mbps
- Execute from STA2:
- *iperf -u -s -i 1*
- Execute from AP:
- *iperf -c 192.168.1.11 -u -b 1000M -i 1 -t 10000*
- 818 Mbps

The sample output for the features described previously is as follows.

```
root@OpenWrt:/# Iwpriv ath0 vap_txrx_st_rst 14
root@OpenWrt:/# Iwpriv ath0 vap_txrx_st_rst 16
root@OpenWrt:/# Iwpriv ath0 vap_txrx_st_rst 17
root@OpenWrt:/# iwpriv ath0 vap_txrx_stats 17

root@OpenWrt:/#
[ 313.875164] mu_sch_nusers_2 : 0
[ 313.880224] mu_sch_nusers_3 : 0
[ 313.884161] mu_mpdus_queued_usr0 : 0
[ 313.887628] mu_mpdus_queued_usr1 : 0
[ 313.891502] mu_mpdus_queued_usr2 : 0
[ 313.895970] mu_mpdus_queued_usr3 : 0
[ 313.914745] mu_mpdus_failed_usr0 : 0
[ 313.918181] mu_mpdus_failed_usr1 : 0
[ 313.922399] mu_mpdus_failed_usr2 : 0
[ 313.925960] mu_mpdus_failed_usr3 : 0

root@OpenWrt:/# iwpriv ath0 vap_txrx_stats 16

[ 376.291502] TX_SELFGEN Info:
[ 376.291533]
root@OpenWrt:/# [ 376.295220] su_ndpa : 0
[ 376.299875] su_ndp : 0
[ 376.303092] su_bar : 0
[ 376.306404] su_cts2self : 0
[ 376.309622] su_ndpa_err : 0
[ 376.313402] su_ndp_err : 0
[ 376.316338] mu_ndpa : 0
[ 376.319618] mu_ndp : 0
[ 376.322930] mu_brpoll_1 : 0
[ 376.326116] mu_brpoll_2 : 0

root@OpenWrt:/# iwpriv ath0 vap_txrx_stats 14
```

```

[ 395.338394] TXBF Sounding Info:
root@OpenWrt:/# [ 395.340706] Sounding User 1 : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 395.347735] Sounding User 2 : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 395.353670] Sounding User 3 : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 395.358856] CBF 20 :IBF 0, EBF 0, MU 0
[ 395.362667] CBF 40 :IBF 0, EBF 0, MU 0
[ 395.366166] CBF 80 :IBF 0, EBF 0, MU 0
root@OpenWrt:/# Iwpriv ath0 vap_txrx_st_rst 14
root@OpenWrt:/# Iwpriv ath0 vap_txrx_st_rst 16
root@OpenWrt:/# Iwpriv ath0 vap_txrx_st_rst 17

root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 1
[ 309.801312] User poistions: 0 0 0 0
[ 309.801343] User poistions: 0 0 0 0
[ 309.801343] User poistions: 0 0 0 0
[ 309.801343] User poistions: 0 0 0 0

root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 2
[ 319.478131] User poistions: 1 1 0 1
[ 319.478131] User poistions: 0 0 1 0
[ 319.478131] User poistions: 0 0 1 0
[ 319.478131] User poistions: 0 0 0 0

root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 2
[ 1464.297032] MUMIMO tx for this peer 82851

root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 1
[ 1469.293033] MUMIMO tx for this peer 83299

root@OpenWrt:/# iwpriv ath0 rst_mu_tx_count 1
root@OpenWrt:/# iwpriv ath0 rst_mu_tx_count 2
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 2
[ 408.637675] MUMIMO tx for this peer 1069
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 1
[ 412.422461] MUMIMO tx for this peer 2833

root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 3
[ 442.374476] Invalid value. Check valid AID values with wlanconfig athN
list sta

root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 0
[ 444.646922] Invalid AID. List valid AID with: wlanconfig athN list sta
Interface doesn't accept private ioctl...
get_mu_tx_count (8BE0): Invalid argument

```

### 10.28.7.6 Sample results

- iwpriv athx get\_mu\_tx\_count AID
 

```
[ 1729.364792] MUMIMO tx for this peer 3762
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 39
[ 1729.397407] MUMIMO tx for this peer 3840
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 40
[ 1729.433708] MUMIMO tx for this peer 3752
root@OpenWrt:/#
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 41
```

```
[ 1729.505060] MUMIMO tx for this peer 4078
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 42
root@OpenWrt:/# [ 1729.538113] MUMIMO tx for this peer 4010
iwpriv ath0 get_mu_tx_count 43
[ 1729.567041] MUMIMO tx for this peer 4138
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 44
[ 1729.598937] MUMIMO tx for this peer 3807
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 45
[ 1729.633770] MUMIMO tx for this peer 4398
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 46
[ 1729.678038] MUMIMO tx for this peer 4039
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 47
[ 1729.707685] MUMIMO tx for this peer 3693
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 48
[ 1729.738956] MUMIMO tx for this peer 3853
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 49
[ 1729.771040] MUMIMO tx for this peer 4006
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 50
[ 1729.800937] MUMIMO tx for this peer 4639
root@OpenWrt:/#
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 51
[ 1729.863792] MUMIMO tx for this peer 0
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 52
[ 1729.896376] MUMIMO tx for this peer 3863
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 53
[ 1729.925710] MUMIMO tx for this peer 4002
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 54
[ 1729.959044] MUMIMO tx for this peer 0
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 55
[ 1729.988816] MUMIMO tx for this peer 4096
root@OpenWrt:/# iwpriv ath0 get_mu_tx_count 56
[ 1730.019868] MUMIMO tx for this peer 4026
```

- **iwpriv athx get\_mu\_peer\_pos AID**

```
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 1
[ 1580.729647] User poistions: 0 0 0 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 1
[ 1585.490690] User poistions: 0 0 0 0
root@OpenWrt:/#
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 2
3
[ 1588.109028] User poistions: 0 0 0 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 4
[ 1589.783848] User poistions: 0 0 0 0
```

```
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 5
[ 1591.182692] User poistions: 0 2 2 0
[ 1591.182692] User poistions: 0 1 0 0
[ 1591.182692] User poistions: 0 1 0 0
[ 1591.182692] User poistions: 0 1 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 6
[ 1592.582661] User poistions: 0 0 0 0
[ 1592.582661] User poistions: 0 0 0 0
[ 1592.582692] User poistions: 0 0 0 0
[ 1592.582692] User poistions: 0 0 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 7
[ 1594.034645] User poistions: 2 2 3 0
[ 1594.034645] User poistions: 1 1 0 1
[ 1594.034645] User poistions: 0 1 0 1
[ 1594.034645] User poistions: 0 1 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 8
[ 1595.492158] User poistions: 3 3 3 1
[ 1595.492189] User poistions: 1 1 1 1
[ 1595.492189] User poistions: 0 1 1 1
[ 1595.492189] User poistions: 0 1 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 9
[ 1596.863980] User poistions: 0 0 0 0
[ 1596.863980] User poistions: 0 0 0 0
[ 1596.863980] User poistions: 0 0 0 0
[ 1596.863980] User poistions: 0 0 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 10
[ 1598.393033] User poistions: 1 1 0 3
[ 1598.393033] User poistions: 2 2 1 0
[ 1598.393033] User poistions: 1 0 1 0
[ 1598.393033] User poistions: 0 0 1
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 11
[ 1600.668228] User poistions: 2 0 1 2
[ 1600.668259] User poistions: 3 2 0 1
[ 1600.668259] User poistions: 1 0 0 1
[ 1600.668259] User poistions: 0 0 1
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 12
[ 1602.442549] User poistions: 3 1 1 3
[ 1602.442549] User poistions: 3 2 1 1
[ 1602.442580] User poistions: 1 0 1 1
[ 1602.442580] User poistions: 0 0 1
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 13
[ 1603.452358] User poistions: 0 2 2 2
[ 1603.452358] User poistions: 2 3 0 0
[ 1603.452358] User poistions: 1 1 0 0
[ 1603.452358] User poistions: 0 1 1
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 14
[ 1604.438269] User poistions: 1 3 2 3
[ 1604.438269] User poistions: 2 3 1 0
[ 1604.438269] User poistions: 1 1 1 0
[ 1604.438300] User poistions: 0 1 1
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 15
root@OpenWrt:/# [ 1605.644142] User poistions: 2 2 3 2
[ 1605.644142] User poistions: 3 3 0 1
[ 1605.644142] User poistions: 1 1 0 1
[ 1605.644142] User poistions: 0 1 1
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 16
```

```

[ 1607.551608] User poistions: 0 0 0 0
[ 1607.551608] User poistions: 0 0 0 0
[ 1607.551608] User poistions: 0 0 0 0
[ 1607.551608] User poistions: 0 0 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 17
root@OpenWrt:/# \[ 1609.248641] User poistions: 3 3 3 3
[ 1609.248641] User poistions: 3 3 1 1
[ 1609.248672] User poistions: 1 1 1 1
[ 1609.248672] User poistions: 0 1 1
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 18
[ 1610.793158] User poistions: 0 0 0 0
[ 1610.793158] User poistions: 0 0 2 2
[ 1610.793158] User poistions: 2 2 0 0
[ 1610.793158] User poistions: 1 0 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 20
root@OpenWrt:/# [ 1618.715963] User poistions: 2 0 1 0
[ 1618.715963] User poistions: 1 0 2 3
[ 1618.715963] User poistions: 2 2 0 1
[ 1618.715963] User poistions: 1 0 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 21
[ 1620.689315] User poistions: 3 1 1 1
[ 1620.689315] User poistions: 1 0 3 3
[ 1620.689347] User poistions: 2 2 1 1
[ 1620.689347] User poistions: 1 0 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 22
[ 1622.302936] User poistions: 0 2 2 0
[ 1622.302936] User poistions: 0 1 2 2
[ 1622.302967] User poistions: 2 3 0 0
[ 1622.302967] User poistions: 1 1 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 22
root@OpenWrt:/# [ 1623.521680] User poistions: 0 2 2 0
[ 1623.521680] User poistions: 0 1 2 2
[ 1623.521680] User poistions: 2 3 0 0
[ 1623.521712] User poistions: 1 1 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 23
[ 1624.502624] User poistions: 1 3 2 1
[ 1624.502624] User poistions: 0 1 3 2
[ 1624.502624] User poistions: 2 3 1 0
[ 1624.502655] User poistions: 1 1 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 24
[ 1626.529678] User poistions: 2 2 3 0
[ 1626.529678] User poistions: 1 1 2 3
[ 1626.529678] User poistions: 2 3 0 1
[ 1626.529709] User poistions: 1 1 0
root@OpenWrt:/# iwpriv ath0 get_mu_peer_pos 25
[ 1629.959231] User poistions: 3 3 3 1
[ 1629.959262] User poistions: 1 1 3 3
[ 1629.959262] User poistions: 2 3 1 1
[ 1629.959262] User poistions: 1 1 0

```

#### ■ iwpriv athx vap\_txrx\_stats 14

```

root@OpenWrt:/# iwpriv ath01 vap_txrx_stats 14
[ 4540.097415] Sounding User 3 : root@OpenWrt:/# 20Mhz 0, 40Mhz 0, 80Mhz
0
[ 4540.108949] CBF 20 :IBF 0, EBF 0, MU 0
[ 4540.112772] CBF 40 :IBF 0, EBF 0, MU 0

```

```

[ 4540.116420] CBF 80 :IBF 0, EBF 2333, MU 0
[ 4540.120441]
[ 4540.120441]
[ 4540.123501] TXBF Sounding Info:
[ 4540.126489] Sounding User 1   : 20Mhz 0, 40Mhz 0, 80Mhz 2368
[ 4540.132376] Sounding User 2   : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 4540.137853] Sounding User 3   : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 4540.143489] CBF 20 :IBF 0, EBF 0, MU 0
[ 4540.147146] CBF 40 :IBF 0, EBF 0, MU 0
[ 4540.150871] CBF 80 :IBF 0, EBF 2340, MU 0
[ 4540.154867]
[ 4540.154867]
iwpriv ath02 vap_txrx_stats 14
root@OpenWrt:/# iwpriv ath03 vap_txrx_stats 14
[ 4540.170605] TXBF Sounding Info:
root@OpenWrt:/# [ 4540.178649] Sounding User 1   : 20Mhz 0, 40Mhz 0, 80Mhz
2378
[ 4540.184808] Sounding User 2   : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 4540.190352] Sounding User 3   : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 4540.195907] CBF 20 :IBF 0, EBF 0, MU 0
[ 4540.199632] CBF 40 :IBF 0, EBF 0, MU 0
[ 4540.203366] CBF 80 :IBF 0, EBF 2349, MU 0
[ 4540.207357]
[ 4540.207357]
[ 4540.210317] TXBF Sounding Info:
[ 4540.213595] Sounding User 1   : 20Mhz 0, 40Mhz 0, 80Mhz 2353
[ 4540.219265] Sounding User 2   : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 4540.224806] Sounding User 3   : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 4540.230364] CBF 20 :IBF 0, EBF 0, MU 0
[ 4540.234094] CBF 40 :IBF 0, EBF 0, MU 0
[ 4540.237830] CBF 80 :IBF 0, EBF 2321, MU 0
[ 4540.241887]
[ 4540.241887]
iwpriv ath04 vap_txrx_stats 14
iwpriv ath06 vap_txrx_stats 14
[ 4540.248562] TXBF Sounding Info:
[ 4540.253564] Sounding User 1   : root@OpenWrt:/# iwpriv ath05 vap_txrx_
stats 14
20Mhz 0, 40Mhz 0, 80Mhz 2297
root@OpenWrt:/# iwpriv ath06 vap_txrx_stats 14
[ 4540.264953] Sounding User 2   : root@OpenWrt:/# 20Mhz 0, 40Mhz 0, 80Mhz
0
[ 4540.274629] Sounding User 3   : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 4540.280104] CBF 20 :IBF 0, EBF 0, MU 0
[ 4540.283837] CBF 40 :IBF 0, EBF 0, MU 0
[ 4540.287573] CBF 80 :IBF 0, EBF 2280, MU 0
[ 4540.291655]
[ 4540.291655]
[ 4540.302386] TXBF Sounding Info:
[ 4540.304506] Sounding User 1   : 20Mhz 0, 40Mhz 0, 80Mhz 2352
[ 4540.310329] Sounding User 2   : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 4540.315884] Sounding User 3   : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 4540.321517] CBF 20 :IBF 0, EBF 0, MU 0
[ 4540.325163] CBF 40 :IBF 0, EBF 0, MU 0
[ 4540.328894] CBF 80 :IBF 0, EBF 2309, MU 0
[ 4540.332959]

```

```
[ 4540.332959]
[ 4540.335853] TXBF Sounding Info:
[ 4540.339007] Sounding User 1 : 20Mhz 0, 40Mhz 0, 80Mhz 2350
[ 4540.344784] Sounding User 2 : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 4540.350448] Sounding User 3 : 20Mhz 0, 40Mhz 0, 80Mhz 0
[ 4540.355898] CBF 20 :IBF 0, EBF 0, MU 0
[ 4540.359632] CBF 40 :IBF 0, EBF 0, MU 0
[ 4540.363366] CBF 80 :IBF 0, EBF 2320, MU 0
[ 4540.367366]
[ 4540.367366]
root@OpenWrt:/# iwpriv ath0 txrx_fw_stats 14
[ 1312.431271] TXBF Sounding Info:
root@OpenWrt:/# [ 1312.433770] Sounding User 1 : 20Mhz 0, 40Mhz 4, 80Mhz
45318
[ 1312.440799] Sounding User 2 : 20Mhz 0, 40Mhz 0, 80Mhz 12261
[ 1312.446735] Sounding User 3 : 20Mhz 0, 40Mhz 0, 80Mhz 12
[ 1312.452639] CBF 20 :IBF 0, EBF 0, MU 0
[ 1312.456357] CBF 40 :IBF 0, EBF 4, MU 0
[ 1312.459856] CBF 80 :IBF 71, EBF 31828, MU 22631
[ 1312.464323]
[ 1312.464323]
```

■ [iwpriv athx vap\\_txrx\\_stats 16](#)

```
root@OpenWrt:/# iwpriv ath0 vap_txrx_stats 16
[ 4567.749722] TX_SELFGEN Info:
[ 4567.749722]
[ 4567.753324] su_ndpa : 2364
[ 4567.756624] su_ndp : 2361
[ 4567.760182] su_bar : 0
[ 4567.763600] su_cts2self : 0
[ 4567.768186] su_ndpa_err : 12
[ 4567.771676] su_ndp_err : 31
[ 4567.774946] mu_ndpa : 0
[ 4567.778238] mu_ndp : 0
[ 4567.781665] mu_brpoll_1 : 0
[ 4567.785013] mu_brpoll_2 : 0
[ 4567.788310]
[ 4567.788310]
root@OpenWrt:/# iwpriv ath01 vap_txrx_stats 16
[ 4567.795133] TX_SELFGEN Info:
[ 4567.795133]
[ 4567.798480] su_ndpa : 2357
[ 4567.802271] su_ndp : 2356
[ 4567.806983] su_bar : 0
[ 4567.810281] su_cts2self : 0
[ 4567.813799] su_ndpa_err : 11
[ 4567.817674] su_ndp_err : 17
[ 4567.821054] mu_ndpa : 0
[ 4567.824548] mu_ndp : 0
[ 4567.828248] mu_brpoll_1 : 0
[ 4567.831708] mu_brpoll_2 : 0
[ 4567.835546]
[ 4567.835546]
root@OpenWrt:/# iwpriv ath02 vap_txrx_stats 16
root@OpenWrt:/# [ 4567.849209] TX_SELFGEN Info:
[ 4567.849209]
```

```

[ 4567.852726] su_ndpa      : 2365
[ 4567.856372] su_ndp       : 2360
[ 4567.859931] su_bar       : 0
[ 4567.863446] su_cts2self   : 0
[ 4567.867145] su_ndpa_err   : 13
[ 4567.870517] su_ndp_err    : 16
[ 4567.874097] mu_ndpa      : 0
[ 4567.877653] mu_ndp       : 0
[ 4567.880933] mu_brpoll_1   : 0
[ 4567.884436] mu_brpoll_2   : 0
[ 4567.888150]
[ 4567.888150]

root@OpenWrt:/# iwpriv ath0 txrx_fw_stats 16
[ 1315.099500] TX_SELFGEN Info:
[ 1315.099500]
root@OpenWrt:/# [ 1315.103280] su_ndpa      : 32879
[ 1315.108278] su_ndp       : 32836
[ 1315.111777] su_bar       : 9
[ 1315.115089] su_cts2self   : 77094
[ 1315.118712] su_ndpa_err   : 182
[ 1315.122305] su_ndp_err    : 1047
[ 1315.125741] mu_ndpa      : 12203
[ 1315.129490] mu_ndp       : 12186
[ 1315.133052] mu_brpoll_1   : 11721
[ 1315.136676] mu_brpoll_2   : 11
[ 1315.140018] mu_bar_1     : 38033
[ 1315.143798] mu_bar_2     : 34
[ 1315.147328] mu_cts2self   : 29935
[ 1315.150859] mu_ndpa_err   : 58
[ 1315.154233] mu_ndp_err    : 434
[ 1315.157669] mu_brp1_err   : 918
[ 1315.161105] mu_brp2_err   : 0
[ 1315.164511]
[ 1315.164511]

```

■ iwpriv athx vap\_txrx\_stats 17

```

iwpriv ath015 vap_txrx_stats 17[ 1295.144954] TX_MU Info:
[ 1295.144954]
[ 1295.148016] mu_sch_nusers_2      : 0
[ 1295.151858] mu_sch_nusers_3      : 0
[ 1295.155732] mu_mpdus_queued_usr0  : 0
[ 1295.159512] mu_mpdus_queued_usr1  : 0
[ 1295.163448] mu_mpdus_queued_usr2  : 0
[ 1295.167322] mu_mpdus_queued_usr3  : 0
[ 1295.170946] mu_mpdus_failed_usr0  : 0
[ 1295.174757] mu_mpdus_failed_usr1  : 0
[ 1295.178631] mu_mpdus_failed_usr2  : 0
[ 1295.182443] mu_mpdus_failed_usr3  : 0
[ 1295.186316]
[ 1295.186348]
[ 1295.189128] TX_MU Info:
[ 1295.189128]
[ 1295.193189] mu_sch_nusers_2      : 260
[ 1295.197032] mu_sch_nusers_3      : 0
[ 1295.200906] mu_mpdus_queued_usr0  : 44309

```

```

[ 1295.205217] mu_mpdus_queued_usr1   :      0
[ 1295.208934] mu_mpdus_queued_usr2   :  33696
[ 1295.213152] mu_mpdus_queued_usr3   :      0
[ 1295.216932] mu_mpdus_failed_usr0   :      52
[ 1295.220743] mu_mpdus_failed_usr1   :      0
[ 1295.224617] mu_mpdus_failed_usr2   :  1252
[ 1295.228803] mu_mpdus_failed_usr3   :      0
[ 1295.232521]
[ 1295.232521]
[ 1295.235551] TX_MU Info:
[ 1295.235551]
[ 1295.239362] mu_sch_nusers_2       :    684
[ 1295.243392] mu_sch_nusers_3       :      0
[ 1295.247110] mu_mpdus_queued_usr0   :      0
[ 1295.251046] mu_mpdus_queued_usr1   :  42746
[ 1295.255326] mu_mpdus_queued_usr2   :  29921
[ 1295.260356] mu_mpdus_queued_usr3   :  73752
[ 1295.263980] mu_mpdus_failed_usr0   :      0
[ 1295.267822] mu_mpdus_failed_usr1   :    724
[ 1295.271602] mu_mpdus_failed_usr2   :    261
[ 1295.275632] mu_mpdus_failed_usr3   :    359
[ 1295.279537]
[ 1295.279568]
[ 1295.282536] TX_MU Info:
[ 1295.282536]
[ 1295.286379] mu_sch_nusers_2       :    483
[ 1295.290440] mu_sch_nusers_3       :      0
[ 1295.294158] mu_mpdus_queued_usr0   :  80381
[ 1295.298406] mu_mpdus_queued_usr1   :      0
[ 1295.302186] mu_mpdus_queued_usr2   :  41613
[ 1295.306466] mu_mpdus_queued_usr3   :  37449
[ 1295.310590] mu_mpdus_failed_usr0   :    1846
[ 1295.314620] mu_mpdus_failed_usr1   :      0
[ 1295.318431] mu_mpdus_failed_usr2   :    513
[ 1295.322461] mu_mpdus_failed_usr3   :    897
[ 1295.326398]

```

## 10.28.8 MU-MIMO data path enhancements overview

Changes to the peer-based flow scheduling in the data path enable better Tx performance (downlink) in scenarios such as MU-MIMO multi-client, SU-clients, air-time-fairness (ATF), and wireless multi-media (WMM). The WLAN target subsystem works with a limited amount of allocated memory and it allocates a limited set of descriptors in the transmit data path. In general, this set of descriptors is tuned for peak performance of the WLAN PHY. However, this set of descriptors is not optimally used when certain use-cases are exercised.

The use cases that need efficient management of the descriptor resources include:

- MU-MIMO performance with large number of clients

In the existing design, since the MU grouping algorithms run in the firmware and the host downloads the descriptors to the target, which may not be correlated to the TxOP of the next MU group that the firmware scheduler decides, there is lack of optimality of descriptor utilization in the target.

- ### ■ Weak sister problem

When there are multiple data flows to different peers with different PHY rate capabilities, the limited descriptors in the firmware are consumed by a peer that has a lower data rate (e.g. 802.11n client) thereby hogging another peer that has a higher data rate.

- #### ■ Multi-client performance

The aggregate throughput of the system with multiple peers should not degrade sharply as the number of active clients are gradually increased.

- ATF

ATF algorithm run in the firmware and the host downloads descriptors to target for unrelated clients whose TXOP may not be scheduled by the firmware.

- ### ■ Video-over-wireless

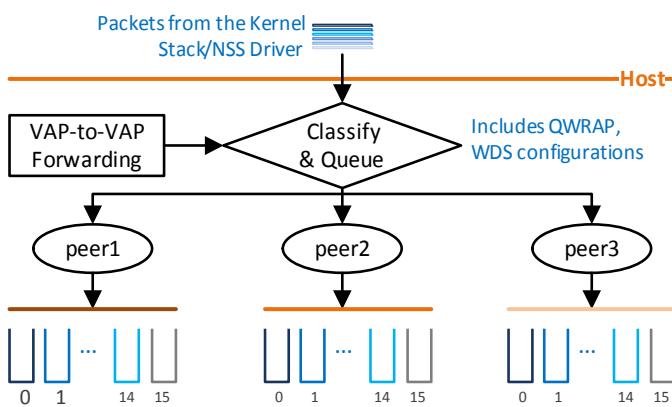
- WMM

- ## ■ Peer-caching with large number of active clients

The new architecture ensures efficient management of the descriptor resources for all these instances by using the host processor and memory to accumulate the backlog while the firmware processor schedules and pulls the descriptors in to its limited pool in order to maximize the throughput and the user experience.

### 10.28.8.1 High-level design

The following figure shows a high level depiction of the peer-based flow schedule design. The host inspects the packets coming in from the kernel stack/ NSS driver/ WiFi-VAP-VAP forwarding module, then classifies the packets and stores them in per-peer/TID queues. For unicast frames, peer lookup is done based on the destination MAC address. Multi-cast/broadcast frames are assigned to AP-BSS peer queue. TID is extracted from DSCP/QoS information in 802.11 QoS/ IPv4/v6 headers.



**Figure 10-10** High-level peer-based slow schedule design

- The host driver periodically updates the queue depth of the various per-peer, per-TID queues at a pre-negotiated memory location that was exchanged between the host and the target.

- The pre-fetch scheduler running on the target firmware sends an HTT fetch message to the host to de-queue the packets from the host queue and send them to the target. This message indicates the number of packet descriptors and packet bytes to be downloaded per peer, per-TID, to the target for this fetch.
- This HTT\_T2H\_MSG\_TYPE\_FETCH message is a variable length message carrying the number of records in the message as the first byte in the message. This byte is followed by the records each of which comprises the details related to peer, TID, number of descriptors to download and number of packet bytes.
- For each fetch message from the target, the host de-queues the packets to target and sends a response message of type HTT\_H2T\_MSG\_TYPE\_FETCH RESP back to the target. This acts as acknowledgment (ACK/NACK) to the target and has the information about the number of requests that have been made from the target, by the host.

## Configuration

Peer-based flow scheduling is enabled through a compile time PEER\_FLOW\_CONTROL flag, which is enabled by default. These runtime parameters are provided for performance tuning based on the system memory availability and specific performance requirements:

- Global threshold limit on the total number of buffers (for all the TID queues) that can be queued on host for flow control.

```
iwpriv wifi0 fc_buf_max <value>
```

- Maximum threshold for per-peer/per-TID queue size

```
iwpriv wifi0 fc_q_max <value>
```

- Minimum threshold for per-peer/per-TID queue size

```
iwpriv wifi0 fc_q_min <value>
```

### 10.28.8.2 Software/firmware architecture

#### With global flow control

- The host receives packets from the stack.
  - If FW has descriptors available, it passes packets through the HTT\_MSDU messages
  - If no descriptors are available, the host code queues the packets
- FW receives the packets and de-classifies them based on the destination address and QoS.
- Packets are then segregated into FW queues ( TIDQ)
- Once the firmware queues are full, an implicit backpressure is exercised to the host, triggering it to start queueing packets.
- The scheduler inside FW round robins across TIDQ and creates a PPDU for transmission.
- At the end of the transmission, the HTT\_COMPLETION message is sent to the host to indicate which packets have been transmitted..
- HTT\_COMPLETION informs the host about the availability of freed descriptors.. Host starts sending the queued packets to FW.

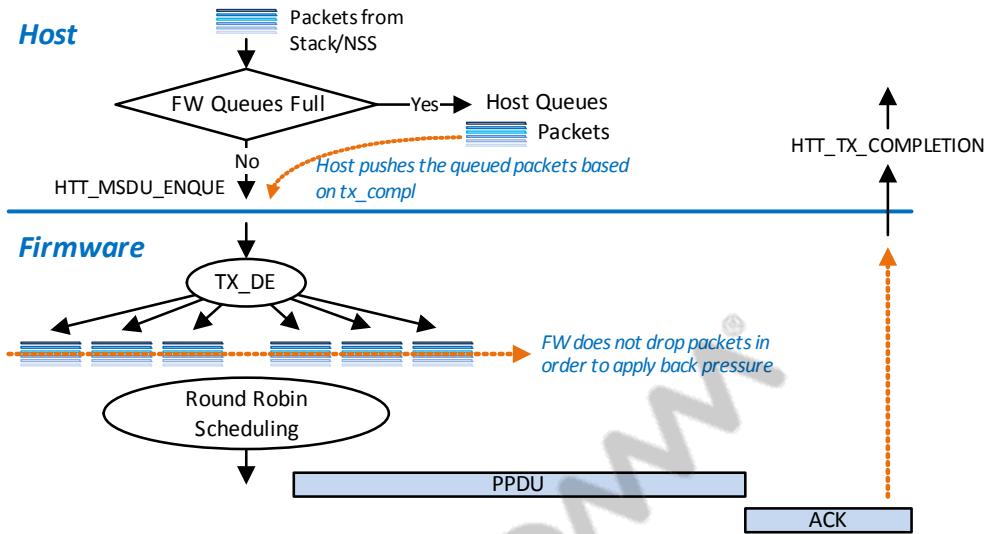
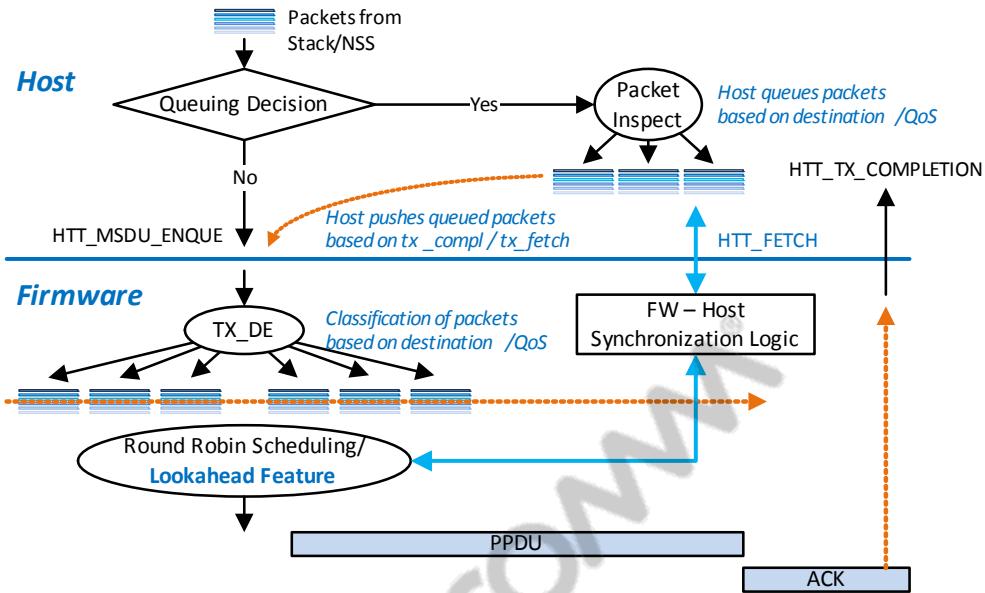


Figure 10-11 SW/FW architecture with global flow control

### With lookahead feature

- The host receives packets from stack. If FW has descriptors available, it passes packets through the HTT\_MSU messages.
  - If the number of descriptors used < a threshold, the host directly pushes packets to FW.
  - If the number of descriptors used > a threshold, host code does a packet inspection and queues packets to the a per peer queue in the host.
- FW periodically gets a snapshot of the host-queue via a mirrored host-queue state transferred using CE.
- FW looks through the host-queue state and sends HTT fetch indication messages to the host to de-queue packets from a specific queue in the host.
- FW receives the packets and stores them in TIDQs. The host sends a HTT fetch response message once the download finishes.
- FW then constructs the PPDU and pushes the packet to HW.



**Figure 10-12 SW/FW architecture with lookahead feature**

### 10.28.9 Enhanced MU-MIMO debugging statistics

To debug the MU MIMO problems in an efficient and optimal manner, the following additional debug commands are introduced:

- Tx Rx Firmware Stats 6 to maintain separate statistical counters for number of MU packets transmitted.
- Per-peer counter to determine total number of times the peer is prevented from MU transmission due to sounding failures.

#### Track MU packets transmitted using the iwpriv tx\_rx\_fw\_stats 6 command

A new field is added to track the MU packets transmitted at each MCS. Separate counters are maintained for SU & MU. Also the total number of Packets (SU + MU) is displayed. The following is the sample input and output for this command:

`iwpriv <interface> txrx_fw_stats 6`

When MU traffic is being run, the following output is displayed:

```

[ 1038.864737] TX Rate Info:
[ 1038.869289] MCS counts (0..9): 0, 0, 0, 0, 0, 3, 43, 12, 0, 42842 << Total Packets
(SU+MU)
[ 1038.876155] MCS counts SU (0..9): 0, 0, 0, 0, 0, 3, 24, 1, 0, 1 << SU Packets
[ 1038.882414] MCS counts MU (0..9): 0, 0, 0, 0, 0, 0, 19, 11, 0, 42841 << MU Packets
[ 1038.889607] SGI counts (0..9): 0, 0, 0, 0, 0, 0, 1, 0, 42842
[ 1038.895977] NSS counts: 1x1 21513, 2x2 21387, 3x3 0 4x4 0
[ 1038.901385] BW counts: 20MHz 0, 40MHz 0, 80MHz 42900, 160MHz 0

```

```
[ 1038.907839] Preamble (O C H V) counts: 34, 0, 0, 43072
[ 1038.912614] STBC rate counts (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
[ 1038.920943] LDPC Counts: 42900
[ 1038.923463] RTS Counts: 0
[ 1038.926878] Ack RSSI: 46
```

When SU traffic is being run, the following output is displayed:

```
root@OpenWrt:/# iwpriv ath0 txrx_fw_st_rst 0xffff
root@OpenWrt:/# iwpriv ath0 txrx_fw_stats 6
[ 2119.377214] TX Rate Info:
[ 2119.379095] MCS counts (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 10172
[ 2119.385340] MCS counts SU (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 10172
[ 2119.393372] MCS counts MU (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0
[ 2119.400672] SGI counts (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 10172
[ 2119.406979] NSS counts: 1x1 0, 2x2 10172, 3x3 0 4x4 0
[ 2119.412905] BW counts: 20MHz 0, 40MHz 0, 80MHz 10172, 160MHz 0
[ 2119.419692] Preamble (O C H V) counts: 9, 0, 0, 10185
[ 2119.423764] STBC rate counts (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0
[ 2119.431332] LDPC Counts: 10172
[ 2119.435220] RTS Counts: 0
[ 2119.437613] Ack RSSI: 39
```

### Track MU Transmission Blacklist counter per peer

If there is continuous sounding Failures for a Peer, that peer will be blacklisted for a duration of 5 seconds from MU Transmission. Currently there are no counters to know how long / how many times this peer has been blacklisted from MU Transmission. This information is maintained for all the Peers.

A new iwpriv command, iwpriv <athX> mu\_blklist\_cnt <AID>, is added with the AID as an argument. This command is used to display the counter that shows the number of times a Peer has been blacklisted from MU Transmission due to Sounding Failures.

The following is the sample input and output:

```
root@OpenWrt:/# iwpriv ath0 mu_blklist_cnt 1
[ 2382.474478] Number of times MU tx blacklisted 21
root@OpenWrt:/# iwpriv ath0 mu_blklist_cnt 2
[ 2384.233901] Number of times MU tx blacklisted 3
```

When an invalid AID is entered, the following is the message displayed:

```
root@OpenWrt:/# iwpriv ath0 mu_blklist_cnt 10
```

[ 3640.503809] Invalid AID value.

## 10.28.10 Enhanced MU-MIMO for scoring metrics and MU grouping

The main objectives of the MU MIMO Enhancements are to address the following problems in the existing implementation:

1. Scoring Metrics Deficiencies

Scoring takes into account only the MAC efficiency and not the actual throughput over the air (OTA).

2. Ignored Effects of MU Grouping

For each remote device (Peer), track the PER for the following configurations:

- a. SU

- b. MU2 (Sum of Partners NSS = 1) – Tracked with Single MU2 PER Table

- c. MU2 (Sum of Partners NSS = 2) – Tracked with Single MU2 PER Table

- d. MU3 (Sum of Partners NSS = 2) – Tracked with Single MU3 PER Table

- e. MU3 (Sum of Partners NSS = 3) – Tracked with Single MU3 PER Table

With Single PER Table maintained for different MU2 / MU3 Configuration not considering the Partner NSS results in PER fluctuation depending on the NSS chosen for transmission.

### 10.28.10.1 Scoring Metrics Computation

The following scoring metric is considered for the actual throughput achieved:

$N = \text{num\_users}$ ,  $n = \text{user idx}$ ,  $m = \text{transmit mode (SU/MU2/MU3)}$

$\text{tx\_air\_time\_m} = \text{ppdu\_data\_tx\_time\_m} + \text{tx\_overhead\_time\_m}$

$\text{user\_n\_tx\_data\_bytes\_m} = \text{ppdu\_data\_tx\_time\_m} * \text{tx\_data\_rate\_m\_n}$

$\text{score\_m} = \text{throughput\_m} = \frac{\sum_{n=1}^N \text{user\_n\_tx\_data\_bytes\_m}}{\text{tx\_air\_time\_m}}$

The score now takes into account the sounding / transmit overheads, as well as the actual amount of data that can be transmitted during this TXOP due to the TX PHY rate. This allows selection of the best transmit mode that should optimize throughput.

Choosing SU/MU2/MU3 from Tx Candidate Table is modified to use actual throughput instead of MAC Efficiency.

### 10.28.10.2 MU Probing and Additional PER Tables

When MU  $\text{Nss} == \text{Ntx}$ , PER is higher compared to  $\text{Nss} < \text{Ntx}$ .

Track them in separate PER Tables due to different MCS rates.

MU2 Mode => 2 PER Tables namely {MU2 - 2ss, 3ss}, {MU2 - 4ss}

MU3 Mode => 2 PER Tables namely {MU3 - 3ss}, {MU3 - 4ss}

Within MU2 & MU3 Modes, choose the mode that gives the best throughput.

Keeping the PER Tables updated by probing each of the RC MODE.

### **Limitations**

- For QCA9980 2x2 Solutions, MU MIMO Enhancements are not enabled due to Lower DRAM. This is needed to accommodate additional PER Tables. MU is less than SU in 360 orientation tests in some orientations.
- In 360 orientation tests, it is possible to achieve MU Gain  $\geq 0.9$  in all orientations and not 1.0.
- MU-MIMO enhancements in QCA9984 VHT160 mode are not supported.

## **10.29 Latency optimization**

To service different test cases and use cases that require tighter scheduling latency requirement, it is necessary to optimize the transmit path to reduce the average turnaround to ensure that no visible latency is measured for these testcases and usecases.

### **Terminology**

|          |                           |
|----------|---------------------------|
| TXOP     | Transmission Opportunity  |
| PPDU     | PLCP Protocol Data Unit   |
| SEQ-CTRL | Transmit Sequence Control |

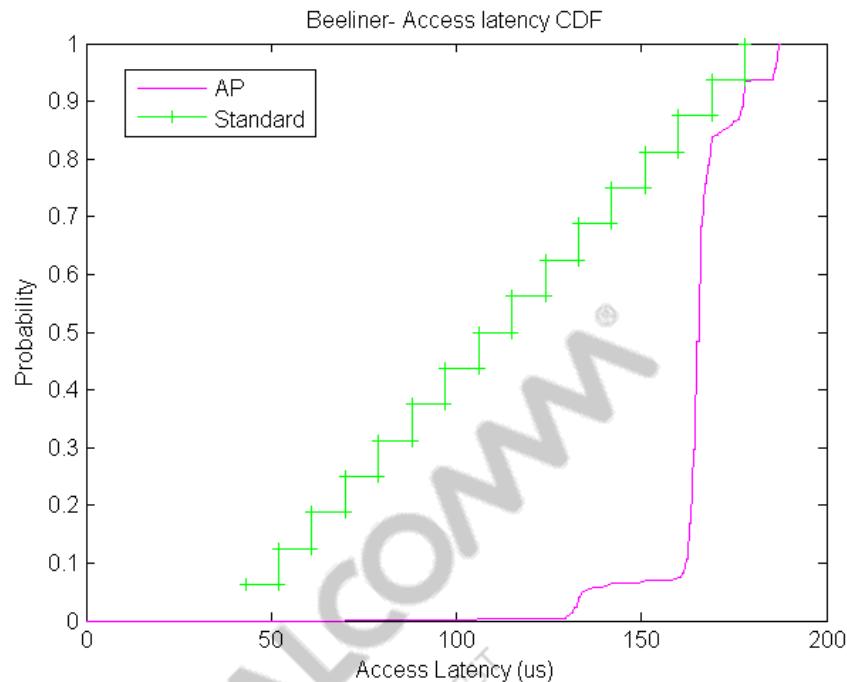
### **References**

The overall latency requirement is broken down into two, one is within TXOP and another is across TXOP.

In QCA9980 family of chips, the latency requirement is resolved within a TXOP by splitting the work with ISR that handles the latency path that requires the transmit path to re post within 50 usec. However, the second part where the time taken to post the next SEQ-CTRL or TXOP is not completely addressed.

### **10.29.1 Problem statement**

There are two test-case we are trying to solve one is the standalone test-case that we use to measure the latency and another is with co channel transmission from other APs.



In the above slide the average QCA9980 latency is in the order of 150 usec, if we could meet within 50 usec we could have got a latency result, this would mean in case of co-channel use case where we are coexisting with another AP we will get less chances to win the medium.

Test cases prove we do have a latency concern but however these may not be the only one we should be concerned about, these test cases were run with just UDP traffic, but with TCP these latency will be less pronounced due to the overlap of receive time that could be used to prepare the next transmission.

Other test cases that we would need to focus on is scheduling latency between two different users, scheduling latency with bursty/intermittent traffic, this is a typical case for small window size TCP. In most of these use cases meeting latency may not help always, in some scenarios we are better off delaying the schedule so we could aggregate more, however in this feature request the focus is on reducing the latency and also how this will positively effect the test cases run by our customers.

## 10.29.2 Objectives

Given that QCA9980 family of chips are already in production, reduction of latency is implemented without rearchitecting the entire code and also to show improvement in test cases run by our customers, with the serious memory limitation for both code and data we should have an option on having different configuration for different chips.

These are the test cases that are addressed in the generation of chips:

- Single user UDP latency
- Single user UDP latency with co-channel transmission from other AP's
- Single user, and single stream TCP performance.

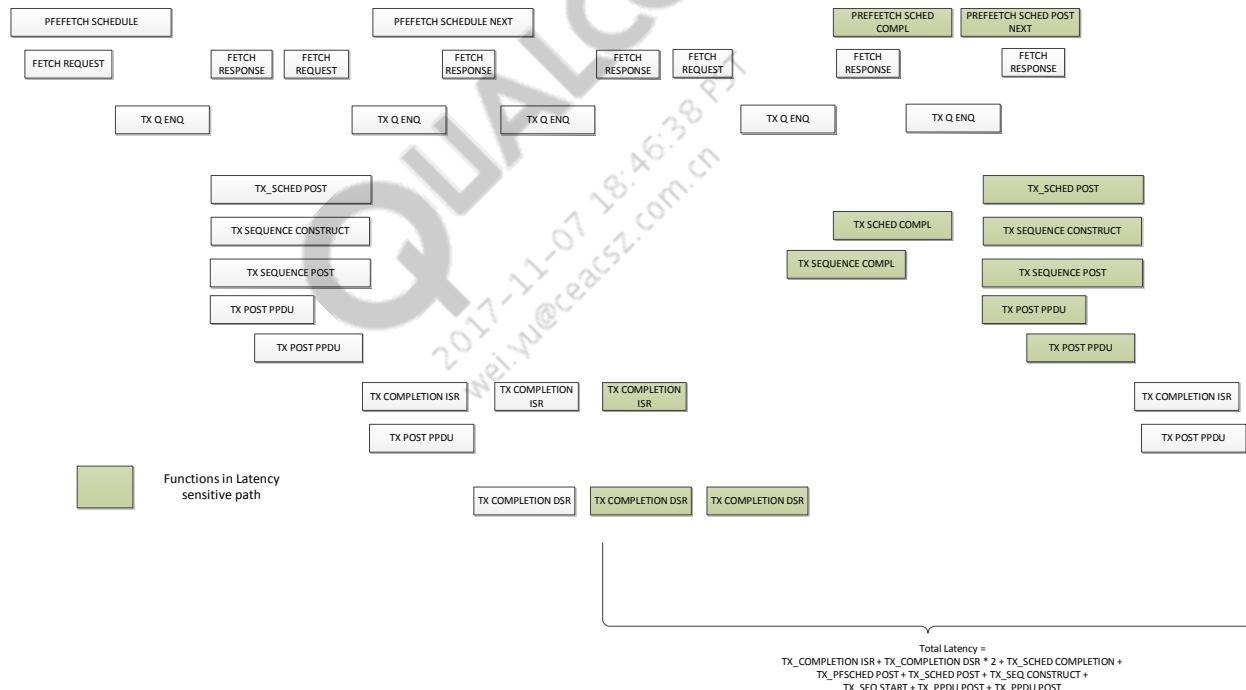
- Single user and single stream and small window size performance.

These are the test cases that are not covered in this generation for practical reasons:

- Latency for MU (*For MU-MIMO in general improving this latency has very less impact, infact we do add latency in MU path to get better performance*).
- Latency of schedules across different users
- Memory constraints

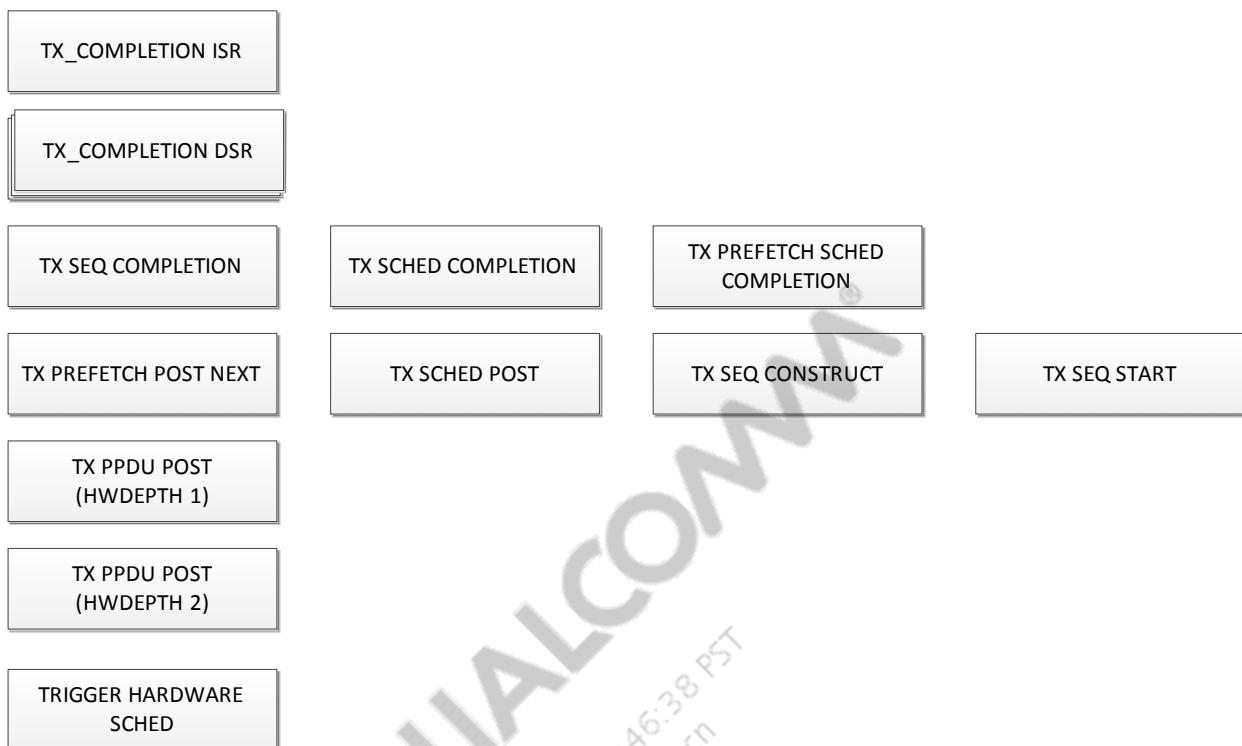
### 10.29.3 Design

This section describes the current design and modules in the latency sensitive path and how different latencies are reduced, while some latency cannot be removed due to the memory and other practical reasons (time and effort).



**Figure 10-13 Tx Post/completion timing diagram**

The preceding figure illustrates the different functions in the latency sensitive path.



**Figure 10-14 Function in the Latency sensitive path**

The overall latency optimization means the following phases of implementation

1. Optimize the above functions.
2. Pipeline the next schedule in DSR so we can turnaround quickly by bypassing certain function, have two sched/seq\_ctrl pending.
3. Pipeline the next schedule/seq\_ctrl in ISR.
4. Cache the last schedule/sequence control so we can eliminate majority of the work that needs to be done when posting the next sequence.
5. Have an infinite schedule, this can be helpful for fully backed up traffic as well as for memory constrained system like QCA9980 where we don't have DRAM so we cannot pipeline two schedule commands.
6. Post the next sequence from transmit completion interrupt (TX\_ISR)
7. For Steering and data sequence (non sounding sequence) we could potentially trigger the hardware after posting the first PPDU, instead of triggering after the second PPDU.

## 10.30 FILS Discovery frames

This section describes the host-level implementation of Fast Initial Link Setup (FILS) Discovery Frame for WFA Optimized Connectivity Experience (OCE). FILS protocol defined in 802.11ai specification is helpful to reduce the connection delay WLAN initial connection delay without

compromising on security. FILS Discovery Frame is a public action frame transmitted by an AP that indicates its capability information to support FILS.

FILS Discovery frame will be transmitted if following conditions are met.

- Bursted beacon mode is enabled.
- FILS Discovery interval must be greater than or equal to 20 TUs.
- FILS Discovery interval must be less than Beacon interval.
- Beacon interval must be divisible by FILS Discovery interval.

### 10.30.1 Enable/disable FILS Discovery Frame support

The iwpriv command, ‘enable\_fils’, is modified to include an extra argument, which provides the flexibility to configure the period in time units (TUs) of FILS discovery (FD) frames.

```
iwpriv <vap> enable_fils <1/0> <period>
```

- vap—VAP name
- 1/0—1 to enable, 0 to disable
- Period—Time period between each FD frames in TUs.

The following are examples of FD period configuration:

- #iwpriv ath0 enable\_fils 1 20—Command to enable FILS FD frame with period set to 20 TUs
- #iwpriv ath0 enable\_fils 0 20—Command to disable FILS FD frame

|   |   |  |
|---|---|--|
| <pre>enable_fils &lt;1   0&gt; &lt;period&gt; g_enable_fils</pre> | <pre>iwpriv athX enable_fils &lt;1   0&gt; &lt;period&gt; iwpriv athX g_enable_fils</pre> | <p>Enable FILS capability and also configure the period or time units (TUs) of FILS discovery (FD) frames. To configure the period (TUs) of FD frames, enter the <code>iwpriv &lt;vap&gt; enable_fils &lt;1/0&gt; &lt;period&gt;</code> command.</p> <p><b>Examples:</b></p> <p><code>iwpriv ath0 enable_fils 1 20</code>—Command to enable FILS FD frame with period set to 20 TUs<br/> <code>iwpriv ath0 enable_fils 0 20</code>—Command to disable FILS FD frame</p> <p>To retrieve the configured period of FD frames, enter the <code>iwpriv athX g_enable_fils</code> command.</p> <p>FILS Discovery frame will be transmitted if following conditions are met.</p> <ul style="list-style-type: none"> <li>■ Bursted beacon mode is enabled.</li> <li>■ FILS Discovery interval must be greater than or equal to 20 TUs.</li> <li>■ FILS Discovery interval must be less than Beacon interval.</li> <li>■ Beacon interval must be divisible by FILS Discovery interval.</li> </ul> |
|---|---|--|

## 10.30.2 UCI commands to configure FILS Discovery Frames

The support to enable FILS through UCI is extended to add support for FD frames. Setting the option, ‘ieee80211ai’, triggers the iwpriv command mentioned in the previous section. A new option, ‘fils\_fd\_period’, is added to allow a user to configure the FD frame period. If this option is not mentioned, 20 TUs is considered as the default period. A sample UCI configuration to enable FILS is as follows:

```
uci set wireless.@wifi-iface[0].ssid='FILS-5G'
uci set wireless.@wifi-iface[0].encryption='wpa2+ccmp'
uci set wireless.@wifi-iface[0].wpa_group_rekey='3600'
uci set wireless.@wifi-iface[0].key='12345678'
uci set wireless.@wifi-iface[0].ieee80211ai=1
uci set wireless.@wifi-iface[0].fils_realm=example.com
uci set wireless.@wifi-iface[0].fils_cache_id='1234'
uci set wireless.@wifi-iface[0].add_sha256='1'
uci set wireless.@wifi-iface[0].erp_send_reauth_start=1
uci set wireless.@wifi-iface[0].erp_domain=example.com
uci set wireless.@wifi-iface[0].auth_server_port=1812
uci set wireless.@wifi-iface[0].server=192.168.1.35
uci set wireless.wifi0.disabled='0'
uci set wireless.wifi0.channel=36
uci set wireless.@wifi-iface[0].own_ip_addr=192.168.1.1
uci set wireless.@wifi-iface[0].dhcp_server=192.168.1.55
uci set wireless.@wifi-iface[0].dhcp_rapid_commit_proxy=1
uci commit wireless
wifi load
```

Note that the default period is 20TUs. To explicitly set fd\_period, the following command must be issued. This command sets fd\_period to 100TUs.

```
uci set wireless.@wifi-iface[0].fils_fd_period=100
uci commit wireless
wifi load
```

## 10.30.3 WMI command to enable FD frames

A new WMI command, WMI\_ENABLE\_FILS\_CMDID, is added to indicate firmware about enable/disable this feature. On receiving the iwpriv command1, WMI\_ENABLE\_FILS\_CMDID is sent to the firmware. This WMI command takes vdev\_id and fd\_period as arguments. After receiving a disable command, the fd\_period is set to 0. If fd\_period is set as 0, firmware considers it as an indication to disable the feature. For non-zero values in fd\_period, it is treated as an indication to enable the feature with the value in the fd\_period as the period for FD frames.

## 10.30.4 SWFDA event

In the firmware, a hardware timer is set up to generate a Software FILS Discovery Alert (SWFDA) event (similar to SWBA) every few ms before the actual FILS Discovery (FD) Frame transmission. Host registers an event handler to receive this event, WMI\_HOST\_SWFDA\_EVENTID.

The buffer for FILS discovery is allocated during VAP up time. The buffer is also initialized and appropriate fields are set during this time. The following fields are currently set:

- Action category—0x4 (Public Action).
- Action—0x22 (FILS Discovery).
- FD Control Subfield (2 bytes)—SSID length set to 0x3, Short SSID indication set (Bit 6)
- Time Stamp (8 bytes)—Set to 0 by host. Timestamp is set by the firmware.
- Beacon Interval (2 bytes)—Beacon interval of AP. Default is 0x64.
- Short SSID (4 bytes)—Short SSID calculated from the SSID.
- Length (1 byte)—Length of remaining field in FD information field. Currently, it is set as 0.
- FILS indication element

The buffer is re-initialized when SSID changes.

### 10.30.5 Send FD buffer frame to the firmware

Host sends the WMI command, WMI\_PDEV\_SEND\_FD\_CMDID, after the SWFDA event is received and FILS Discovery Frame is ready. The FILS Discovery (FD) buffer prepared in the above section is passed in this WMI command. The timestamp and the sequence number are inserted by the firmware/hardware.

## 10.31 Support for dynamically enabling and disabling burst

With burst enabled, it is observed in certain network deployments with multi-clients, such as more than 10 clients, key performance indicator (KPI) degradation occurs with client phones. These drops in KPIs are not observed with burst disabled. To specify enabling and disabling of burst dynamically per interface, an iwpriv command is introduced. Enter the iwpriv athX dyn\_burst 1 command to enable burst. Alternatively, enter the iwpriv athX dyn\_burst 0 command to enable burst. Enter the iwpriv athX get\_dyn\_burst command to retrieve the configured setting of dynamic burst.

After this feature is enabled, firmware senses the medium, and based on failure, firmware disables the burst for a certain duration. The burst can be enabled, based on threshold limit.

In the WMI command path, changes added to enable/disable dynamic auto burst feature. A new WMI command, WMI\_DYN\_AUTO\_BURST\_ENABLE\_CMDID, is added to enable/disable the dynamic auto burst support. This setting is stored it in \_wal\_pdev structure for further use in the data path.

|                            |  |   |   |  |
|----------------------------|--|---|---|--|
| dyn_burst<br>get_dyn_burst | iwpriv athX dyn_burst <1   0><br>iwpriv athX get_dyn_burst | N | Y | <p>Enable (1) or disable (0) burst dynamically per interface. With burst enabled, it is observed in certain network deployments with multi-clients, such as more than 10 clients, key performance indicator (KPI) degradation occurs with client phones. These drops in KPIs are not observed with burst disabled.</p> <p>Enter the iwpriv athX get_dyn_burst command to retrieve the configured setting of dynamic burst.</p> |
|----------------------------|--|---|---|--|

# 11 IEEE 802.11 Features

This chapter describes the different IEEE 802.11 specifications and protocols that are supported, such as 802.11ac, 802.11n, 802.11w Protected Management Frames (PMF), and 802.11ad (WiGig).

## 11.1 802.11ac (Very High Throughput) Features

VHT features supported on QCA9880 based products include all mandatory and some optional features defined in the *IEEE P802.11ac/D3.1 specification*.

- 5 GHz support
- VHT AP and STA connectivity
- 20/40/80 MHz channel width
- MCS 0-9
- 1, 2 and 3 Spatial streams
- Short Guard Interval
- TX and RX STBC
- VHT A-MPDU delimiter for RX/TX for Single MPDU
- A-MPDU RX/TX
- A-MSDU RX/TX
- CCA on secondary
- Static and Dynamic Bandwidth Signaling
- Bandwidth Signaling using RTS and CTS

**Table 11-1 802.11ac features**

| Protocol Capability  | References   |
|--|--|
| VHT Capabilities Element   | 8.4.2.160.1 (VHT Capabilities Element Structure)   |
| STA capabilities signaled via Probe requests, (Re)Association requests                   | 8.3.3.9 (Probe request frame format)<br>8.3.3.5 (Association Request format)<br>8.3.3.7 (Reassociation request format)                               |
| STA/BSS capabilities signaled via Beacons, Probe responses and (Re)Association responses | 8.3.3.2 (Beacon frame format)<br>8.3.3.10(Probe response format)<br>8.3.3.6 (Association response format)<br>8.3.3.8 (Reassociation response format) |

**Table 11-1 802.11ac features**

| <b>Protocol Capability</b>  | <b>References</b>  |
|---|--|
| Signaling of VHT operation  | 8.4.2.161 (VHT Operation element)  |
| VHT BSS Operation. Use of primary 20MHz, secondary 20MHz and secondary 40MHz channels.  | 10.39.1 (Basic VHT BSS functionality)<br>10.39.2 (Channel selection methods for VHT BSS)<br>10.39.3 (Scanning requirements for VHT BSS)<br>9.3.2.5a (VHT RTS Procedure)<br>9.3.2.6 (CTS procedure) |
| Country, Power Constraint and transmit power control elements in beacons and probe responses  | 8.4.2.10 (Country Element)<br>8.4.2.17 (Power Constraint)  |
| CCA on primary 20MHz, CCA on secondary 20MHz and secondary 40MHz  | 10.39.5 (NAV assertion in a VHT BSS)   |
| VHT Transmit Power Envelope element in beacon and probe response frames   | 8.4.2.164 (VHT Transmit Power Envelope Element)  |
| Transmission and Reception of Channel Wrapper element and procedures in conjunction with Channel Switch announcement and extended channel switch announcement, Wide Band Channel Switch Element | 10.39.1 (Basic VHT BSS functionality)  |
| Reception of Operating Mode Notification Frame and Operating Mode Notification Element  | 8.5.23.4 (Operating Mode Notification frame format)<br>(Operating Mode Notification Element)   |

### 11.1.1 Initialization

At initialization the target will issue a WMI READY event to the host, which will include the following capabilities for use with VHT mode of operation:

- VHT enabled/disabled
- Max AMPDU length (3895, 7991, or 11454 bytes)
- RX LDPC support status
- SHORT\_GI
- TX STBC support status
- RX STBC – Number of spatial streams supported
- +HTC-VHT capability support status (reception of HT control field in VHT format)
- Maximum A-MPDU Length exponent (AMPDU length pre-EOF padding) – 0-7
- Supported MCS (number of Spatial streams supported and range MCS 0-7/MCS0-8/MCS0-9) for RX and TX
- Max data rate the STA can receive and transmit (Mbps) for VHT
- VHT Basic MCS rate set (MCS 0-7 with 1 SS is mandatory)

The host on receipt of this event will store this data for future use.

## 11.1.2 Configuration

The following attributes can be configured:

- Mode (B/G/A/NAHT,NGHT, VHT)
- Channel width (20/40/80) MHz
- Spatial streams (NSS) (1-3)
- SGI (400 or 800) ns
- VHT MCS index (0-9)
- TX STBC enable/disable
- LDPC enable/disable

| Commands                            | Syntax (iwpriv)                        | Description   | Sample cfg commands for radio0 |
|-------------------------------------|--|---|--------------------------------|
| Setting VHT mode                    | iwpriv <interface> mode <desired_mode> | New desired modes include 11ACVHT20, 11ACVHT40, 11ACVHT80   | cfg -a AP_CHMODE 11ACVHT20     |
| Setting Channel Bandwidth           | iwpriv <interface> chwidth <0-2>       | 0= 20MHz, 1=40MHz, 2=80MHz  | cfg -a AP_CHMODE 11ACVHT80     |
| Setting number of Spatial Streams   | iwpriv <interface> nss <1-3>           | 1= 1SS, 2=2SS, 3=3SS  | cfg -a NSS 2                   |
| Setting VHT MCS Index               | iwpriv ath0 vhtmcs <0-9>               | Use this command along with chwidth and nss command to set fixed VHT rate. Example:<br>iwpriv ath0 chwidth 3<br>iwpriv ath0 nss 3<br>iwpriv ath0 vhtmcs 7 | cfg -a VHT_MCS 7               |
| Enable/Disable LDPC                 | iwpriv ath0 ldpc <0 1>                 | 0=disable 1=enable  | cfg -a LDPC 1                  |
| Enable/Disable TX_STBC              | iwpriv ath0 tx_stbc <0 1>              | 0=disable 1=enable  | cfg -a TX_STBC 1               |
| Enable/Disable Short Guard Interval | iwpriv ath0 shortgi <0 1>              | 0=disable 1=enable  | cfg -a SHORTGI 1               |
| Generate VHT Opmode notification    | iwpriv <interface> opmode_notify 1     | Use 1 for opmode generation.  | Not a persistent command.      |

## 11.1.3 Operation

The UMAC on host constructs all management frames (except for ADDBA, DELBA) with a 802.11 header and sends it to the target. Beacons are sent to the target via WMI messages. All other management frames are sent to the HTT, which forwards them to the target at the appropriate time either as pass by value (or) pass by reference.

The FW (WAL) will setup the sequence number part of the descriptor for the hardware to update the sequence number of the frame. The target maintains the per node/per TID sequence number for management frames.

Action frames such as ADDBA and DELBA are completely handled in the firmware for performance offload reasons.

#### 11.1.4 AP Mode

1. VHT is enabled if the desired PHY mode is set to VH, the channel is VHT capable, and the silicon can support the same (based on the hardware capabilities received via WMI READY).
2. ACS is enhanced to support channel selection procedures as described in section 10.39.2 of the VHT specification. However, a configuration option will be available to force the interface to operate in a desired channel and width.
3. Once the VAP is initialized and operational the AP will
  - Generate Beacons with VHT capability IE and VHT operation IE
  - Parse VHT elements in probe request and generate probe responses
  - Parse association/reassociation requests (VHT capability IEs) and generate association/reassociation responses. The negotiated capabilities and rate sets are propagated to the target.

**NOTE** Te HT IE fields (A-MSDU length, AMPDU length exponent, MCS maps) are adjusted to reflect the values consistent with VHT IE.

4. The target on receipt of WMI messages for operating mode, channel negotiated capabilities and rate will update the associated data structures for use by modules residing on the target.
5. On receipt of deauthentication or disassociation from the STA, the connection manager will tear down the connection.

#### 11.1.5 STA Mode

1. VHT is enabled if the desired PHY mode is set to VHT and the silicon can support it (based on the hardware capabilities received via WMI READY).
2. When a VAP is created with the STA (client) mode and the SSID is specified, the connection management module residing in the UMAC starts scanning the wireless channels to find the infrastructure networks. The client will advertise the VHT Capabilities IE in its probe requests.

**NOTE** The HT IE fields (A-MSDU length, AMPDU length exponent, MCS maps) are adjusted to reflect the values consistent with VHT IE. The STA on receiving beacons/probe responses will parse the VHT elements if present and switch the channel on which the desired AP is present.

3. The client establishes a connection with the desired AP using the Authentication and Association handshake. An association request (with VHT capability IE) will be issued to the

AP of interest and on receipt of an association response, the negotiated capabilities and rate set information will be saved and sent to the target. The connection management maintains continuous connectivity by monitoring for changes in the BSS by examining beacons.

4. On receipt of deauthentication or disassociation from the associated AP, the connection manager will tear down the connection.

## 11.2 160/80+80 MHz Operation

**NOTE** This feature is applicable only for QCA9984 and QCA9888.

The 802.11ac standard requires all devices to support 20, 40, and 80 MHz channel bandwidths in the 5GHz band, with support of 160 MHz channel bandwidth being optional.

QCA9984/QCA9888, which is architecturally derived from AR900B, supports 160 MHz channel bandwidth.

The main bandwidth related QCA9984 feature additions are as follows:

- Support for 2x2 80+80 MHz (up to 2SS)
- Support for 4SS MU in 80 MHz (vs 3SS in QCA9980).
- Support for 2SS MU in 80+80 MHz
- Support for dynamic switching between 80MHz 4x4 and 160/80 + 80 MHz 2x2 on a per packet basis for Tx and Rx.

The main bandwidth related QCA9888 feature additions are as follows:

- Support for 2x2 80+80 MHz (1SS)
- Support for 2SS MU in 80 MHz
- Support for 1SS MU in 80+80 MHz
- Support for dynamic switching between 80 MHz 2x2 and 160/80 + 80 MHz 1x1 on a per packet basis for Tx and Rx.

**NOTE** 160 MHz is treated as a special case of adjacent 80+80. But this has no end-user visible implications.

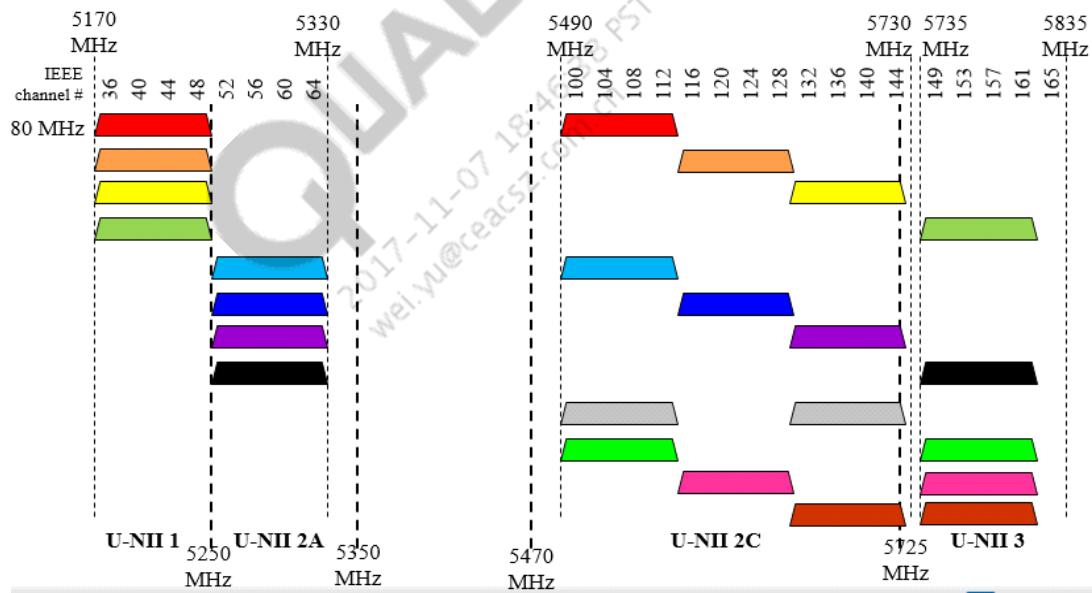
### 11.2.1 Allowed Channels

#### 11.2.1.1 80+80 MHz

80+80 MHz can be formed by combining two non-adjacent 80 MHz channels. Therefore, the difference between the center frequencies of the two segments must be greater than 80. Any channels permitted for 80 MHz can be used in an 80+80 MHz combination, provided the two 80 MHz segments are non-adjacent.

Allowed combinations of frequencies for 80+80 MHz are as follows (subject to any country specific restrictions, and channel 144 support):

- 5210, 5530
- 5210, 5610
- 5210, 5690
- 5210, 5775
- 5290, 5530
- 5290, 5610
- 5290, 5690
- 5290, 5775
- 5530, 5690
- 5530, 5775
- 5610, 5775
- 5690, 5775



**Figure 11-1 Frequencies allowed for 80+80 bandwidth when RegDomain Extension Register Bit = 1**

### 11.2.1.2 160 MHz

Only 2 center frequencies are allowed which are as follows. Further restrictions can apply for some countries:

- 5250
- 5570

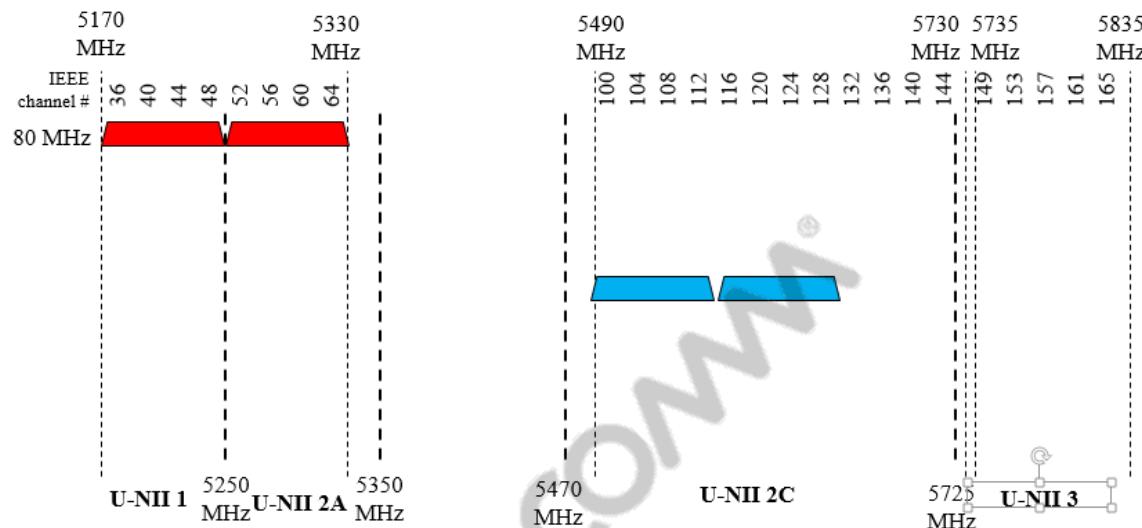


Figure 11-2 Frequencies allowed for 160 MHz bandwidth

### 11.2.1.3 QCA channel 144 support

EEPROM gives information to the host during attach time about which channel bandwidths are supported by the hardware, and additional channelization information.

- RegDomain Extension Register Bit 5 in EEPROM is defined to support FCC CH. 144
  - Bit 5 = 0, do not enable CH.144 for countries in FCC3 and FCC6 domain.
  - Bit 5 = 1, enable CH. 144 for countries in FCC3 and FCC6 domain.

Accordingly, it may or may not have 80MHz band 5650 to 5730 MHz. Hence, this will affect number of pairs we can have for 80+80 channel width. However, this will not affect number of channels allowed for 160.

### 11.2.1.4 Implementation

#### EEPROM support

EEPROM gives information to the host during attach time about which channel bandwidths are supported by the hardware, as well as other channelization information.

ModesAvail is the variable which informs us about the hardware capabilities.

ModesAvail is populated using variable `hal_reg_capabilities.wireless_modes`. `hal_reg_capabilities` is of type `HAL_REG_CAPABILITIES`.

```

typedef struct {
    A_UINT32 eeprom_rd;           // regdomain value specified in EEPROM
    A_UINT32 eeprom_rd_ext;      // regdomain
    A_UINT32 regcap1;            // CAP1 capabilities bit map.
    A_UINT32 regcap2;            // REGDMN EEPROM CAP.
    A_UINT32 wireless_modes;     // REGDMN MODE
    A_UINT32 low_2ghz_chan;
    A_UINT32 high_2ghz_chan;
    A_UINT32 low_5ghz_chan;
    A_UINT32 high_5ghz_chan;
} HAL_REG_CAPABILITIES;

```

hal\_reg\_capabilities is populated by wmi\_service\_ready\_event, which is received by the driver. If 21<sup>st</sup> and 22<sup>nd</sup> (value = 0x7f9001) bits of modesAvail variable are set, that indicates 160 and 80+80 modes are allowed respectively.

### 11.2.1.5 Channel initialization in driver

Driver sets up channel list based on EEPROM information and any supplied country code. EEPROM verification and setup of certain regulatory-related access control data is done by the driver. This driver internal copy of channel list is saved so that we can constrain future requests to these channels.

Initially, depending on the mode, center frequencies are populated into an array of structures. These arrays are vht80\_80\_chans, vht160\_chans and vht80\_chans. Definition of this structure consists of one matrix that will hold frequencies and an integer that keeps count of number of elements in matrix.

```

struct ol_regdmn_vht160_cmn_chan {
    u_int16_t    n_vht160_cmn_ch;
    u_int16_t    vht160_cmn_ch[OL_REGDMN_VHT160_CMN_CHAN_MAX][2];
};

```

This will provide center frequencies for all the modes. In case of 80+80 and 160 (special case of 80+80, with both segments being adjacent), we will have two center frequencies for one entry. These center frequencies are calculated by traversing through 2 GHz and 5 GHz band, respectively. MKK capability check, U1 ODD, U1 EVEN, U2, and MIDBAND bits are checked and accordingly bands to channel list are added or skipped. RD extension bit 5 is checked to allow channel 144 for countries in FCC3 and FCC6 domain. All the 144 channels are skipped when RD bit 5 is equal to 0 or when channel 144 is not allowed as primary channel. In this instance, an array of structure ieee80211\_channel is created, based on the modes supported by the device. This array will contain the following information.

```

struct ieee80211_channel {
    u_int16_t      ic_freq;          /* setting in Mhz */
    u_int32_t      ic_flags;         /* see below */
    u_int8_t       ic_flagext;       /* see below */
    u_int8_t       ic_ieee;          /* IEEE channel number */
    int8_t        ic_maxregpower;    /* maximum regulatory tx power in dBm*/
    int8_t        ic_maxpower;        /* maximum tx power in dBm */
    int8_t        ic_minpower;        /* minimum tx power in dBm */

    u_int8_t ic_regClassId; /* regClassId of this channel */

    u_int8_t ic_antennamax; /* antenna gain max from regulatory */
}

```

```
/* For 80 MHz BSS bandwidth, indicates the channel center frequency index for the 80 MHz
channel on which the VHT BSS operates. For 160 MHz BSS bandwidth, indicates the channel
center frequency index of the 80 MHz channel segment that contains the primary channel. For
80+80 MHz BSS bandwidth, indicates the channel center frequency index for the primary 80 MHz
channel of the VHT BSS. */
u_int8_t ic_vhtop_ch_freq_seg1;
```

```
/* For 160 MHz BSS bandwidth, indicates the channel center frequency index of the 160 MHz
channel on which the VHT BSS operates. For 80+80 MHz BSS bandwidth, indicates the channel
center frequency index of the secondary 80 MHz channel of the VHT BSS. */
u_int8_t ic_vhtop_ch_freq_seg2;
};
```

Based on initial array which contained center frequencies, the allowed primary channels are mapped for any mode, to the center frequencies. The other attributes like ic\_flags, ic\_flagext(DFS), maximum regulatory Tx power, minimum Tx power, antenna gain max from regulatory which are obtained from EEPROM are also filled in this array. There are separate DFS enables for the primary and secondary segment.

The 11ACVHT160 mode has been treated as a special case of 80+80 to take care of attributes like ic\_flags and Tx power for second band. For example, IEEE80211\_CHAN\_DFS flag is set when first band contains any DFS channel but IEEE80211\_CHAN\_DFS\_CFREQ2 flag is set for DFS channel in second band.

Therefore, initially for 11ACVHT160 we have two center frequencies (with difference equal to 80) but later the first center frequency is made equal to average of first and second and second center frequency as zero.

### 11.2.1.6 Information Element modifications

#### VHT capabilities element

The VHT capabilities element contains a number of fields that are used to advertise VHT capabilities of a VHT STA.

- Bit 6 in VHT capabilities information indicates short GI support for the reception of packets transmitted with TXVECTOR parameters FORMAT equal to VHT and CH\_BANDWIDTH equal to CBW160 or CBW80+80. This information is received from wmi\_service\_ready\_event-> vht\_cap\_info.
- The explicitly set bandwidth capabilities for STA in case of QCA9984/QCA9886 when it is greater than 80 MHz (Bit 2 and Bit 3).
  - Set to 0 if the STA does not support either 160 or 80+80 MHz
  - Set to 1 if the STA supports 160 MHz
  - Set to 2 if the STA supports 160 MHz and 80+80 MHz

Since this bandwidth information depends on chip as well as user configuration of bandwidth on STA, we need to handle this dynamically on host. This is helpful when we parse this information on AP side to decide at what bandwidth STA gets connected to AP.

### 11.2.1.7 VHT operation element

The operation of VHT STAs in the BSS is controlled by the HT Operation element and the VHT Operation Element.

- Octet 1 in VHT operation information defines the BSS operating channel width.
  - Set to 0 for 20 MHz or 40 MHz operating channel width.
  - Set to 1 for 80 MHz operating channel width.
  - Set to 2 for 160 MHz operating channel width.
  - Set to 3 for 80+80 MHz operating channel width.
- Values in the range 4 to 255 are reserved.
- Octet 3 indicates the channel center frequency index of the 80 MHz channel of frequency segment 1 on which VHT BSS operates. Reserved otherwise.

### 11.2.1.8 VHT Transmit Power Envelope element

The VHT Transmit Power Envelope element conveys the local maximum transmit power for various transmission bandwidths.

Modify Local Maximum Transmit Power Count in Transmit Power Information field from 2 to 3 in the case of QCA9984. The Local Maximum Transmit Power Count subfield indicates the number of Local Maximum Transmit Power elements for X MHz fields (where X = 20, 40, 80, or 160/80+80) minus 1 in the VHT Transmit Power Envelope element.

### 11.2.1.9 Mode Negotiation

#### AP operating in 11ACVHT80\_80 mode

- If STA mode is 11ACVHT80\_80, STA will associate in mode 11ACVHT80\_80 and will operate on the same channels as AP.
- If STA mode is 11ACVHT160, STA will associate in mode 11ACVHT80. This is because continuous 80+80 operation cannot be provided to STA when AP itself is operating on discontinuous 80+80 channel width.

### 11.2.1.10 AP operating in 11ACVHT160 mode

- If STA mode is 11ACVHT80\_80, STA will associate in mode 11ACVHT160 and will operate on the same channels as AP. This is because if STA is configured in mode 11ACVHT80\_80, it is expected to associate in mode less than or equal to 11ACVHT80\_80, but AP here is operating on 11ACVHT160.
- If STA mode is 11ACVHT160, STA will associate in mode 11ACVHT160.

### 11.2.1.11 Configuration

#### Checking for allowed channels

Execute the below command

```
wlanconfig athX list chan
Channel 36 : 5180   Mhz 11na C CU V VU V80- 42 V160- 50
Channel 40 : 5200   Mhz 11na C CL V VL V80- 42 V160- 50
Channel 44 : 5220   Mhz 11na C CU V VU V80- 42 V160- 50
Channel 48 : 5240   Mhz 11na C CL V VL V80- 42 V160- 50
Channel 52 : 5260 *~ Mhz 11na C CU V VU V80- 58 V160- 50
Channel 56 : 5280 *~ Mhz 11na C CL V VL V80- 58 V160- 50
Channel 60 : 5300 *~ Mhz 11na C CU V VU V80- 58 V160- 50
Channel 64 : 5320 *~ Mhz 11na C CL V VL V80- 58 V160- 50
Channel 100 : 5500 *~ Mhz 11na C CU V VU V80-106 V160-114
Channel 104 : 5520 *~ Mhz 11na C CL V VL V80-106 V160-114
Channel 108 : 5540 *~ Mhz 11na C CU V VU V80-106 V160-114
Channel 112 : 5560 *~ Mhz 11na C CL V VL V80-106 V160-114
Channel 116 : 5580 *~ Mhz 11na C CU V VU V80-122 V160-114
Channel 120 : 5600 *~ Mhz 11na C CL V VL V80-122 V160-114
Channel 124 : 5620 *~ Mhz 11na C CU V VU V80-122 V160-114
Channel 128 : 5640 *~ Mhz 11na C CL V VL V80-122 V160-114
Channel 132 : 5660 *~ Mhz 11na C CU V VU
Channel 136 : 5680 *~ Mhz 11na C CL V VL
Channel 140 : 5700 *~ Mhz 11na C V
Channel 149 : 5745   Mhz 11na C CU V VU V80-155
Channel 153 : 5765   Mhz 11na C CL V VL V80-155
Channel 157 : 5785   Mhz 11na C CU V VU V80-155
Channel 161 : 5805   Mhz 11na C CL V VL V80-155
Channel 165 : 5825   Mhz 11na C V
```

Channels marked 'V160' are capable of participating in 160 MHz mode. The center frequency index is provided after "V160-". Channels having the same center frequency index all participate in the same 160 MHz span.

For example, channel 36 can be the primary 20 MHz of a 160 MHz BSS. The center frequency index is 50. Channels 36, 40, 44, 48, 52, 56, 60, 64 participate in the same 160 MHz span. Channel 40 will be the secondary 20 MHz channel. Channels 44 and 48 are part of the secondary 40 MHz segment. Channels 52-64 form the secondary 80 MHz.

Any channels marked V80 can participate in an 80+80 MHz combination, provided the two segments are not adjacent.

### 11.2.1.12 80+80 MHz configuration

#### iwpriv based

Setting mode:

```
iwpriv athX mode 11ACVHT80_80
```

Setting channel:

Since this needs two frequencies for two bands, we split this in two parts.

The first part takes care of the channel center frequency index of the secondary 80 MHz segment. Center frequency can also be given instead of index.

```
iwpriv athX cfreq2 106 or
iwpriv athX cfreq2 5530
```

The second part takes care of the primary 20 MHz channel. From this, center frequency of primary 80 MHz will be calculated in host.

```
iwconfig athX channel 36
ifconfig athX down && ifconfig athX up
```

The sequence should be followed as given above, i.e., first setting mode, then cfreq2, then primary 20 MHz. This allows the driver to carry out the necessary validation checks.

### **UCI based**

```
wifi detect > /etc/config/wireless
uci commit wireless
uci revert -P /var/state wireless
wifi

uci delete wireless.@wifi-iface[0]
uci add wireless wifi-iface
uci set wireless.wifi0.disabled=0
uci set wireless.wifi0.htmode=HT80_80

uci set wireless.@wifi-device[0].hwmode=11ac
uci set wireless.@wifi-iface[0].network=lan
uci set wireless.@wifi-iface[0].mode=ap
uci set wireless.@wifi-iface[0].encryption=none
uci set wireless.@wifi-iface[0].wds=1
uci set wireless.@wifi-iface[0].ssid="OpenWrt"
uci set wireless.@wifi-iface[0].cfreq2=106
uci set wireless.@wifi-iface[0].device=wifi0
uci set wireless.@wifi-device[0].channel=36

uci commit wireless
wifi down; wifi up
```

### **11.2.1.13 160 MHz configuration**

#### **iwpriv based**

Setting mode: iwpriv athX mode 11ACVHT160

Setting channel: User can specify only the primary channel for 160 MHz bandwidth and from this, the center frequency will be calculated automatically within the driver because the primary channel specified will map to only one center frequency.

```
iwconfig athX channel 36
```

**NOTE** Revised signaling for 160/80+80 MHz can be enabled/disabled using the below command. Refer to [Section 11.2.3](#) for more details.

```
iwpriv athX revsig160 <0/1>
0:disable, 1:enable
Default is 1.
```

### UCI based

```
UCI based
wifi detect > /etc/config/wireless
uci commit wireless
uci revert -P /var/state wireless
wifi

uci delete wireless.@wifi-iface[0]
uci add wireless wifi-iface
uci set wireless.wifi0.disabled=0
uci set wireless.wifi0.htmode=HT160
uci set wireless.@wifi-device[0].hwmode=11ac
uci set wireless.@wifi-iface[0].network=lan
uci set wireless.@wifi-iface[0].mode=ap
uci set wireless.@wifi-iface[0].encryption=none
uci set wireless.@wifi-iface[0].wds=1
uci set wireless.@wifi-iface[0].ssid="OpenWrt"
uci set wireless.@wifi-iface[0].device=wifi0
uci set wireless.@wifi-device[0].channel=36
uci commit wireless
wifi down; wifi up
```

**NOTE** Revised signaling for 160/80+80 MHz can be enabled/disabled using the below command. Refer to [Section 11.2.3](#) for more details.

```
uci set wireless.@wifi-device[0].revsig160=<0/1>
0:disable, 1:enable
Default is 1.
```

## 11.2.2 NSS negotiation in 160/80+80MHz modes

QCA9984 supports up to two spatial streams and QCA9886 supports one spatial stream in 80+80 MHz and 160 MHz modes, though QCA9984 supports four streams and QCA9886 supports two streams in 80 MHz and lesser bandwidths. Since 802.11 standard does not support negotiation of NSS as a function of channel bandwidth, it is not possible to advertise separate NSS capabilities for 80 and 160 MHz modes. Also, the number of spatial streams supported and allowed channel bandwidth is further restricted by the chain-mask configured in QCA9984/QCA9886 PHY. Also, user configuration of channel bandwidth will be limited to 80 MHz if the chain-mask configured in PHY does not allow 160 MHz channels. [Table 11-2](#) lists the maximum number of spatial streams supported by QCA9984/QCA9886 for various possible channel bandwidths, based on the chain-mask configured.

**Table 11-2 Maximum NSS based on chain-mask and bandwidth for QCA9984**

| Chain Mask | Max NSS<br>80/40/20 MHz | Max NSS<br>80+80/160 MHz |
|------------|-------------------------|--------------------------|
| 0b1111     | 4                       | 2                        |
| 0b1110     | 3                       | NA                       |
| 0b1101     | 3                       | NA                       |
| 0b1100     | 2                       | NA                       |
| 0b1011     | 3                       | NA                       |
| 0b1010     | 2                       | 1                        |
| 0b1001     | 2                       | 1                        |
| 0b1000     | 1                       | NA                       |
| 0b0111     | 3                       | NA                       |
| 0b0110     | 2                       | 1                        |
| 0b0101     | 2                       | 1                        |
| 0b0100     | 1                       | NA                       |
| 0b0011     | 2                       | NA                       |
| 0b0010     | 1                       | NA                       |
| 0b0001     | 1                       | NA                       |
|            |                         |                          |

**Table 11-3 Maximum NSS based on chain-mask and bandwidth for QCA9886**

| Chain Mask | Max NSS<br>80/40/20 MHz | Max NSS<br>80+80/160 MHz |
|------------|-------------------------|--------------------------|
| 0b0001     | 1                       | NA                       |
| 0b0011     | 2                       | 1                        |

### 11.2.2.1 NSS negotiation between QCA9984/QCA9886 and similar QCA peers

A proprietary IE is proposed to negotiate NSS capability between QCA9984/QCA9886 based APs and clients based on the above limitations. This is required since the 802.11 standard does not provide mapping Rx NSS values to bandwidth. This IE is added to the following management frame subtypes:

- Probe Request/response frame
- Association Request/Response frame
- Beacon frames (can be disabled by an iwpriv command)

The supported number of spatial streams and channel bandwidth is advertised during association exchange based on the following table.

**Table 11-4 QCA9984 table**

|  | QCA9984 VHT Cap/OMN for chain mask:<br>0b1111 (BW, NSSRx) | QCA9984 VHT Cap/OMN for chain masks<br>0b0111, 0b1011, 0b1101, 0b1110 (BW, NSSRx) | QCA9984 VHT Cap / OMN for chain masks<br>0b1100, 0b0101, 0b1010, 0b1001, 0b0110 (BW, NSSRx) | QCA9984 VHT Cap / OMN for chain mask 0b0011 (BW, NSSRx) | QCA9984 VHT Cap / OMN for chain masks 0b0001, 0b0010, 0b0100, 0b1000 (BW, NSSRx) |
|--|---|---|---|---|--|
| <b>QCA9984 Peer 160 MHz, NSS=4</b>     | 160 MHz, 2 / 80MHz, 4 *                                   | 80 MHz, 3   | 160 MHz, 1 / 80 MHz, 2 *  | 80 MHz, 2   | 80 MHz, 1  |
| <b>QCA9984 Peer 160 MHz, NSSTx = 3</b> | 160M Hz, 2 / 80MHz, 3 *                                   | 80 MHz, 3   | 160 MHz, 1 / 80 MHz, 2 *  | 80 MHz, 2   | 80 MHz, 1  |
| <b>QCA9984 Peer 160 MHz, NSSTx = 2</b> | 160 MHz, 2  | 80 MHz, 2   | 160 MHz, 1 / 80 MHz, 2 *  | 80 MHz, 2   | 80 MHz, 1  |
| <b>QCA9984 Peer 160 MHz, NSSTx = 1</b> | 160 MHz, 1  | 80 MHz, 1   | 160 MHz, 1  | 80 MHz, 1   | 80 MHz, 1  |
|  |   |   |   |   |  |

**Table 11-5 QCA9886 Table**

|                                 | QCA9886 VHT Cap/OMN for chain mask:<br>0b0011 (BW, NSSRx) | QCA9886 VHT Cap / OMN for chain masks<br>0b0001, (BW, NSSRx) |
|---------------------------------|---|--|
| QCA9886 Peer 160 MHz, NSS=2     | 160 MHz, 1 / 80MHz, 2 *                                   | 80 MHz, 1  |
| QCA9886 Peer 160 MHz, NSSTx = 1 | 160 MHz, 1  | 80 MHz, 1  |
|                                 |   |  |

**NOTE** Both modes are supported (160 MHz NSS sent in proprietary IE) and can be chosen dynamically by rate-control

The proprietary information element is defined as per the common general format as described in the specified [IEEE802.11-2012] section 8.4.2.28.

| Element ID | Length | Information field |                         |
|------------|--------|-------------------|-------------------------|
| 221        | n      | CU                | Vendor-specific content |

The Vendor specific IE is explained in detail in following table.

| IIEField   | Description   | Length<br>(Byte) | Value  |  |
|------------|---|------------------|--|--|
| Element ID | Vender specific   | 1                | 0x0d(21) as given by IEEE802.11              |  |
| Length     | Length of vender specific IE                                | 1                | 1 byte                                       |  |
| OUI[3]     | Vender Specific OUI   | 3                | 00-0B-7F                                     |  |
| OUI Type   | OUI Type  | 1                | TBD after checking other QCA proprietary IEs |  |
|            |   |                  | Bit[2:0]                                     | Represents value of Rx NSS for VHT 160 MHz |
| OUI Data   | Bit Map description of HT/MHT bandwidth mapping with Rx NSS | 1                | Bit[7:3]                                     | Reserved                                   |

**NOTE** As per information from the Hardware team, the possibility of NSS varying for 80 MHz and below in future designs is very low. Besides, the NSS value for 160 MHz (contiguous) and 80 + 80 MHz (dis-contiguous) will be the same. Hence, we currently provision for a single value for 160 MHz width, and do not consider lower bandwidths.

## WMI Changes

wmi\_peer\_assoc\_complete\_cmd structure requires adding 1 variable to pass bandwidth mapping with Rx NSS to firmware. Following code snippet shows the additional member variables in blue color.

```

/** num spatial streams */
A_UINT32 peer_nss;
/** VHT capabilties of the peer */
A_UINT32 peer_vht_caps;
/** phy mode */
A_UINT32 peer_phymode;
/** VHT capabilties of the peer */
wmi_vht_rate_set peer_vht_rates;

/** BITMAP representing mapping of bandwidths to Rx NSS */
/*      bit[2:0] : Represents value of Rx NSS for VHT 160 MHz
 *      bit[3:30]: Reserved
 *      bit[31]:    MSB(0/1): 1 in case of valid data else all bits
will be set to 0 by host
*/
A_UINT32 bw_rxnss_override;
} wmi_peer_assoc_complete_cmd;

```

**NOTE** Host driver will be sending valid data only for QCA9984 or QCA9886 chipset and when it associates with the peer as QCA9984 or QCA9886. For other combinations, this field (**bw\_rxnss\_override**) will be set to 0 in which case NSS and BW negotiated and exchanged between host/fw using present existing method holds true.

### 11.2.2.2 NSS/BW negotiation with non-QCA9984/QCA9886 peers

For non-QCA stations that support more than two spatial streams in 80+80/160 MHz modes, NSS and bandwidth negation will be based on the VHT capabilities advertised, as specified in [Table 11-6](#). This negotiation is done based on the capabilities advertised in association request and probe response frames received from non-QCA9984/QCA9886 clients and APs respectively.

**Table 11-6 NSS/BW negotiated with non-QCA9984 solutions**

|   | QCA9984 VHT Cap / OMN for chain mask:<br>0b1111<br>(BW, NSSRx) | QCA9984 VHT Cap / OMN for chain masks<br>0b0111, 0b1011,<br>0b1101, 0b1110<br>(BW, NSSRx) | QCA9984 VHT Cap / OMN for chain masks 0b1100,<br>0b0101, 0b1010,<br>0b1001, 0b0110<br>(BW, NSSRx) | QCA9984 VHT Cap / OMN for chain mask<br>0b0011<br>(BW, NSSRx) | QCA9984 VHT Cap / OMN for chain masks 0b0001,<br>0b0010, 0b0100,<br>0b1000<br>(BW, NSSRx) |
|---|--|---|---|---|---|
| <b>3rd Party Peer<br/>160MHz, NSSTx = 4</b> | 80 MHz, 4  | 80 MHz, 3   | 80 MHz, 2   | 80 MHz, 2   | 80 MHz, 1   |
| <b>3rd Party Peer<br/>160MHz, NSSTx = 3</b> | 160 MHz, 2   | 80 MHz, 3   | 80 MHz, 2   | 80 MHz, 2   | 80 MHz, 1   |
| <b>3rd Party Peer<br/>160MHz, NSSTx = 2</b> | 160 MHz, 2   | 80 MHz, 2   | 80 MHz, 2   | 80 MHz, 2   | 80 MHz, 1   |
| <b>3rd Party Peer<br/>160MHz, NSSTx = 1</b> | 160 MHz, 1   | 80 MHz, 1   | 160 MHz, 1  | 80 MHz, 1   | 80 MHz, 1   |

**Table 11-7 NSS/BW negotiated with non-QCA9886 solutions**

|                                    | QCA9886 VHT Cap / OMN for chain mask:<br>0b0011 (BW, NSSRx) | QCA9886 VHT Cap / OMN for chain masks<br>0b0001,<br>(BW, NSSRx) |
|------------------------------------|---|---|
| 3rd Party Peer<br>160MHz, NSSTx= 4 | 80 MHz, 2   | 80 MHz, 1   |
| 3rd Party Peer<br>160MHz, NSSTx= 3 | 80 MHz, 2   | 80 MHz, 1   |
| 3rd Party Peer<br>160MHz, NSSTx= 2 | 80 MHz, 2   | 80 MHz, 1   |
| 3rd Party Peer<br>160MHz, NSSTx= 1 | 160 MHz, 1  | 80 MHz, 1   |

### 11.2.3 Revised 160/80+80 MHz operation signaling

Legacy (80 MHz) clients in the field were determined to experience interoperability issues when valid 160/80+80 MHz related values were present in VHT Operation Information transmitted by 160/80+80 MHz APs. In January 2016, IEEE approved a change titled "VHT160 operation

signaling through non-zero CCFS1" for 160/80+80 MHz signaling towards solving these interoperability issues without having to mandate legacy 80 MHz client driver updates. Please refer to standards publications from IEEE for details of these changes. A brief summary is given below:

VHT Operation Information (present in VHT Operation IE) is changed as follows:

| Field                                      | Deprecated definitions  | Definitions after revision  |
|--|---|---|
| Channel Width                              | Set to 2 for 160 MHz BSS bandwidth.<br><br>Set to 3 for non-contiguous 80+80 MHz BSS bandwidth.   | Set to 0 for 20 MHz or 40 MHz BSS bandwidth (as before).<br><br>Set to 1 for 80 MHz BSS bandwidth (as before).<br><br><b>Set to 1</b> for contiguous 160 MHz BSS bandwidth (new).<br><br><b>Set to 1</b> for non-contiguous 80+80 MHz BSS bandwidth (new).  |
| Channel Center Frequency Segment 0 (CCSF0) | For 160 MHz BSS bandwidth and the Channel Width subfield equal to 2, indicates the channel center frequency index of the 160 MHz channel on which the VHT BSS operates.<br><br>For 80+80 MHz BSS bandwidth and the Channel Width subfield equal to 3, indicates the channel center frequency index for the primary 80 MHz channel of the VHT BSS. | For 80 MHz BSS bandwidth, indicates the channel center frequency index for the 80 MHz channel on which the VHT BSS operates (as before).<br><br>For 160 MHz BSS bandwidth and the Channel Width subfield <b>equal to 1</b> , indicates the channel center frequency index of the <b>80 MHz channel segment that contains the primary channel</b> (new).<br><br>For 80+80 MHz BSS bandwidth and the Channel Width subfield <b>equal to 1</b> , indicates the channel center frequency index for the primary 80 MHz channel of the VHT BSS (relation with channel width is new, the rest of the definition is as before). |
| Channel Center Frequency Segment 1 (CCSF1) | For a 160 MHz BSS bandwidth and the Channel Width subfield equal to 2, this field is set to 0.<br><br>For an 80+80 MHz BSS bandwidth and the Channel Width subfield equal to 3, indicates the channel center frequency index of the secondary 80 MHz channel of the VHT BSS.  | For a 20, 40 or 80 MHz BSS bandwidth, this subfield is set to 0 (as before).<br><br>For a 160 MHz BSS bandwidth and the Channel Width subfield <b>equal to 1</b> , indicates the <b>channel center frequency index of the 160 MHz channel</b> on which the VHT BSS operates (new).<br><br>For an 80+80 MHz BSS bandwidth and the Channel Width subfield <b>equal to 1</b> , indicates the channel center frequency index of the secondary 80 MHz channel of the VHT BSS (relation with channel width is new, the rest of the definition is as before).  |

In essence, the channel width value of 1 previously used only for 80 MHz is now used for 160/80+80 MHz as well, and CCFS1 is used instead to distinguish between 80 vs. 160 vs. 80+80 MHz. Legacy 80 MHz STAs will continue to always see the values of channel width and CCFS0 which they are used to (and tests indicate they ignore CCFS1), thus alleviating the interoperability issues as per tests.

QCA 10.4 code has been changed so that a 160/80+80 MHz AP uses the revised signaling by default. The user can revert to use of legacy signaling at runtime by setting wifi-iface uci option ‘revsig160’ to 0, or ‘iwpriv athX revsig160 0’. Refer to [Section 11.2.1.13](#)

QCA 10.4 code has been changed so that a 160/80+80 MHz non-AP STA can dynamically distinguish between the legacy and revised signaling, and work with either. No user intervention is required for this.

Use of QCA160/80+80 MHz AP implementing revised signaling with QCA 160/80+80 MHz capable STA from older 10.4 based releases using legacy signaling: The STA will associate in 80 MHz. If it is desired to make it connect in 160/80+80 MHz, the user will have to switch to legacy signaling on the AP using command provided above (with the caveat that this might result in interoperability issues with some legacy 80 MHz STAs, and potentially even newer third party 160/80+80 MHz STAs if these do not implement legacy signaling).

**NOTE** Host driver internal channel data structure (ieee80211\_channel) is updated to reflect the revised definitions of cfreq1 and cfreq2.

## 11.3 802.11n Features

### 11.3.1 A-MPDU Aggregation (Peer Feature Negotiation)

#### 11.3.1.1 Protocol

The 802.11n specification defines the protocol for negotiating A-MPDU aggregation parameters with a peer. This negotiation is bi-directional; that is, each peer has to agree to receive A-MPDU aggregates. Each peer that wants to transmit A-MPDUs sends a request to the other side to establish resources for receiving aggregates. If the other side accepts the request, it will respond with a positive acknowledgment, and specify the resources it has available for receiving an aggregate. A-MPDU aggregation is implemented as a sliding window, called a Block ACK Window (BAW), over a sequence number set. Each peer specifies the number of buffers it has available for reordering packets. This number cannot be exceeded by the sender. Note that aggregation is negotiated independently for each TID being used between a pair of peers. The capability exchange and negotiation is termed “ADDBA exchange”.

After the negotiation is completed successfully, the sender can send aggregate MDPU (A-MPDUs) to the receiver.

#### 11.3.1.2 Implementation

The entry point in the LMAC to transmit a data packet is *ath\_tx\_send* (if\_ath.c). In this function, the packet is first checked for an EAPOL ethertype. The aggregation negotiation is not started if a key exchange is expected but not complete, since it is not desirable to introduce a delay in the 802.11i key exchange. If the peer node is capable of handling aggregates, and the packet is not of type EAPOL, an action frame is sent out with an ADDBA\_REQUEST. The data packet that triggered the ADDBA is then processed as a normal un-aggregated packet.

When the peer has received and processed the ADDBA request, it sends a response, which is processed in *ath\_addba\_responseprocess()*. If the response is a success, the Block ACK Window size is stored in the node, and the node is flagged as being capable of transmitting A-MPDUs.

Similarly, when we receive an ADDBA request, if the node is capable of receiving aggregates, a SUCCESS status is returned with the available resources for the Block ACK Window.

The aggregation mode that was setup between peers can be torn down by sending a DELBA message. This message can originate on either peer. If the initiator is the data transmitter, all retried packets in the block ACK window are freed. The remainder are sent out as single packets. If the initiator is the receiver, all received packets in the window are indicated to the stack. Resources allocated to hold the window are preserved in either case.

## 11.3.2 Data Transmit Aggregation

### 11.3.2.1 Queuing packets for Transmit

Aggregation on the transmit side is implemented in the LMAC (Lower MAC) or ath layer. The entry point in the ath layer is the function *ath\_tx\_send*, which starts an ADDBA exchange if needed, and hands the packet down to *ath\_tx\_start()* for transmit preparation. The determination of the rate at which to transmit an aggregate is delayed until the actual point of queuing to hardware. At this point the packet is mapped for DMA to the MAC, and transmit descriptors are allocated for the scatter gather list for the packet. If the peer-node is capable of receiving aggregates, *ath\_tx\_send\_ampdu* is called now.

*ath\_tx\_send\_ampdu()* checks if there is a transmit already in progress, or if there are other packets in the TID queue for the node. It also ensures that the sequence number of the packet is within the window that is expected by the receiver. If these conditions are satisfied, the packet is queued in the TID-specific queue for the node. If not, the packet is sent as a single (non-aggregate). The packet sequence number is added into the Block Ack Window (*ath\_tx\_addto\_baw*). The node-specific rate is obtained from the rate\_control module, which sets up the rate series and the number of tries at each rate (*ath\_buf\_set\_rate*). Finally, the packet is queued to hardware for transmission (*ath\_tx\_txqaddbuf*). There are multiple hardware queues, each with different WMM parameters corresponding to the TID access class of the packet. If the queue is full, the packet is freed back to the protocol stack.

### 11.3.2.2 Transmit Completion

After the packet is transmitted, the hardware indicates the completion for the access class-specific hardware queue. There is an interrupt from the MAC and the packet that was completed is processed for rate control information, and so on. If the packet is part of an aggregate (indicated by the *bf\_isampdu* flag), the acknowledgment from the receiving peer is a bit map with one bit set for each successful transmit. All successful frames are freed back to the OS/protocol. For unsuccessful packets, if the number of transmit retries is not exceeded, the packet is queued back into the node TID queue (see *ath\_tx\_complete\_aggr\_rifs()*); these are then sent out as part of the next aggregate.

### 11.3.2.3 Aggregate Transmission

At the end of handling transmit completion (in *ath\_tx\_edma\_tasklet*), *ath\_txq\_schedule()* is called for the specific tx queue that had the completion. There could be multiple nodes transmitting packets on the same access class. All of these nodes that have packets queued are put on a “waiting for transmit” TID queue. This queue is processed in a round robin fashion, and the next node with packets pending is pulled off this queue. These packets are then packed into an aggregate (*ath\_tx\_sched\_aggr*, which then calls *ath\_tx\_form\_aggr*). Aggregate formation is based on certain rules: the maximum number of bytes that can be in an aggregate is limited by what was negotiated in the ADDBA exchange. Also, the rate control module (*ath\_lookup\_rate*) returns the maximum number of bytes that can be transmitted to this node, at a specific data rate, over a 4 ms period (this is the maximum time that a node is allowed to transmit, also known as the TXOP limit). A list of packets is compiled based on these rules: hardware transmit descriptors are filled out for each packet (taking care to provide information about each scatter-gather element). After creating the list, the chain is provided to hardware by calling *ath\_tx\_txqaddbuf()*.

**NOTE** The aggregate transmission only happens after a non-aggregate has been transmitted, itself after the idle condition. This allows packets to be collected for a specific node-TID queue. The assumption inherent here is that most network traffic tends to be “bursty”; that is, there is usually more than one packet being transmitted between two peers.

### 11.3.2.4 Excessive Retransmissions of MPDUs

Even after rate drop down, it is possible that some MPDUs do not reach the receiver. These are retried by inserting these MPDUs at the head of new aggregate transmissions. When a maximum number of retries has been reached, the sender gives up on these packets and frees them up. In such an event the receiver is informed by a Block ACK Request packet, the purpose of which is to reset the BAW of the receiver. This is an action frame that contains the new sequence number at which the sender will resume transmitting packets. It lets the receiver know that any previous packets will never be resent so that the receiver can move the sliding window forward.

## 11.4 Configure AMPDU per VAP for DA and OL modes

This section describes the support for Aggregated-MAC Packet Data Unit (AMPDU) per VAP for both direct attach (DA) and offload (OL) modes. Until QCA\_Networking\_2016.SPF.4.0 release, AMPDU support was available on the athX interfaces (protocol layer) in OL mode only. Also, this

support was implemented per radio. Starting with QCA\_Networking\_2017.SPF.5.0, the following enhancements are made:

- AMPDU support is per VAP and not per radio
- AMPDU support is extended for DA mode

Because several vendors configure AMPDU per VAP, it is necessary to make this functionality compatible and consistent.

Specific commands are available for AMPDU settings to enable/disable AMPDU and set the number of sub-frames limit.

In DA mode, enter the iwpriv athX ampdu <0/1> command to enable/disable AMPDU per VAP. By issuing the command, the call trace is as follows:

ieee80211\_ucfg\_setparam() with parameter IEEE80211\_PARAM\_AMPDU

- wlan\_set\_param(vap, IEEE80211\_FEATURE\_AMPDU, value)
- IEEE80211\_ENABLE\_AMPDU(ic)/IEEE80211\_DISABLE\_AMPDU(ic) then ic is marked or cleared with the flag IEEE80211\_FEXT\_AMPDU.

In ieee80211\_node\_join(), if IEEE80211\_IS\_AMPDU\_ENABLED(ic) is false, NI is marked with flag IEEE80211\_NODE\_NOAMPDU.

### AMPDU command guidelines

- In DA mode, enter the iwpriv wifiX AMPDU <0/1> command, which is an equivalent command on wifiX interface to disable/enable AMPDU. This command calls ath\_set\_config with parameter ATH\_PARAM\_AMPDU to set sc->sc\_txaggr. Then, sc->sc\_txaggr is used in multiple places.
- In DA mode, enter the iwpriv wifiX AMPDUFrames {1-64} command to set AMPDU maximum frames limit per radio.
- In OL mode, enter the iwpriv athX ampdu {1-64} to set AMPDU subframes limit to target. The call trace for the command is part the same as in DA path but with additional functions. It calls ol\_ath\_vap\_set\_param with parameter IEEE80211\_FEATURE\_AMPDU, to send AMPDU value to target by the function htt\_h2t\_aggr\_cfg\_msg() called by ol\_txrx\_aggr\_cfg().
- In both DA and OL modes, for AMPDU support per VAP, an effective approach is to use “iwpriv athX ampdu {0-64}” to enable/disable AMPDU and set AMPDU subframes limit. The value 0 indicates that AMPDU is disabled and the values 1-64 indicate the AMPDU subframes limit. The radio-level AMPDU setting for DA path that is available using the iwpriv wifiX AMPDU command remains unchanged. The VAP-level setting and radio-level setting are independent of each other. Only when both are enabled in DA mode, AMPDU is activated.
- In offload mode, enter the iwpriv athX ampdu {0-64} command to set the AMPDU parameter per VAP.
- The value 0 indicates AMPDU must be disabled. The values 1-64 denote the AMPDU subframes limit. The value of 1 signifies the presence of only one frame in an AMPDU packet.
- Enter the iwpriv athX get\_ampdu command to retrieve the AMPDU parameters per VAP.

In the DA path, for wlan\_set\_param (vap, IEEE80211\_FEATURE\_AMPDU, value), a change is made to enable and disable AMPDU at VAP level instead of radio level. In ieee80211\_node\_join(), if AMPDU is disabled at VAP level, the new node indicator (ni) is marked as IEEE80211\_NODE\_NOAMPDU. Also, the value is stored in the struct ieee80211vap and struct ath\_vap\_config to be used in the LMAC function ath\_tx\_form\_aggr().

In the offload path, a new enum member, WMI\_VDEV\_PARAM\_AMPDU\_PER\_AC, is defined in WMI\_VDEV\_PARAM. In ol\_ath\_vap\_set\_param(), the value is sent to target by using wmi\_unified\_vdev\_set\_param\_send() with WMI\_VDEV\_PARAM\_AMPDU\_PER\_AC, VAP ID, and AMPDU value to replace the current function of ht\_h2t\_aggr\_cfg\_msg() called by ol\_txrx\_aggr\_cfg().

#### 11.4.1 AMPDU per VAP

In addition to the support for AMPDU per PDEV/SOC on QCA9980 and QCA9880 chipset families, support is introduced for the AMPDU settings per VAP.

Existing HTT message ‘HTT\_H2T\_MSG\_TYPE\_AGGR\_CFG’ is used to enable/disable AMPDU as well as AMSDU. These are used to configure the maximum number of sub frames in an AMPDU/AMSDU. If maximum number of sub frames in AMPDU is <=1, AMPDU is disabled (or) enabled otherwise. The global variable ‘g\_dbg\_subfrms\_max’ is used for this purpose. ‘g\_dbg\_subfrms\_max’ is set to 64 (MAX\_MPDU\_CNT) by default.

A new WMI VDEV param ‘WMI\_VDEV\_PARAM\_AMPDU\_SUBFRAME\_SIZE\_PER\_AC’ is added to pass on the VAP level AMPDU settings to from Host to the Firmware. A new data structure is added in the WAL peer params to store these settings. If the VAP level value is <=1, AMPDU will be considered disabled for the VAP, else AMPDU is enabled for the VAP with the given AMPDU max subframe size. By default the maximum number of subframes will be 64.

In order to provide backward compatibility with the old Host, the HTT message handler for PDEV level settings would be retained. Going forward this old code can be removed. The Host is expected to use either the PDEV level settings or VAP level settings and not both. The existing iwpriv command ‘ampdu’ will be resued for this feature. i.e., earlier this iwpriv command was doing a PDEV level setting and in the proposed changes, it will perform a VAP level setting.

The blockack negotiation size will remain as before in the old design. i.e., there will not be any change in the window size negotiated with the peer. This will be the max value supported by the DUT.

From the host, a VDEV up and down occurs whenever any change of the AMPDU max sub frame settings takes place.

### 11.5 802.11w Protected Management Frames (PMF)

Protected Management Frames is a feature to protect some types of management frames like deauthorization, disassociation and action frames. This feature prevents attackers from sending plain deauthorization/disassociation frames to disrupt or tear down a connection/association. PMF is a Wi-Fi Alliance specification based on IEEE 802.11w

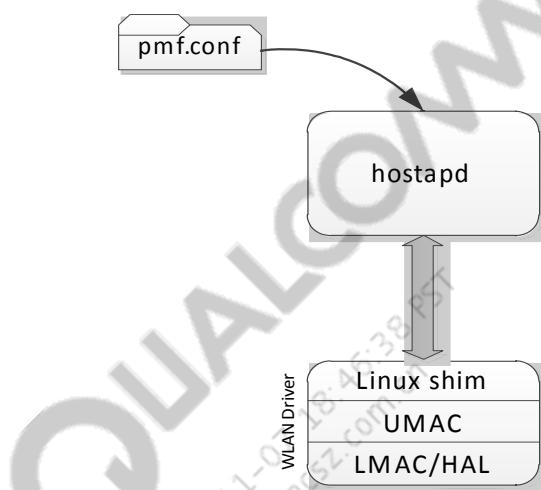
This section describes the PMF implementation on the Access Point (AP) side.

### 11.5.1 Theory of Operation

The AP is configured with SSID, encryption type, authentication type, and PMF capability bit in the RSN IE capability field. Based on those settings, STAs associating with this AP will perform PMF association and get unicast/broadcast/IGTK (Integrated Group Temporal Key) keys. IGTK is used for broadcast, deauthorization, and disassociation frames.

Most of the PMF functionality is handled in hostapd. As part of PMF, new key management algorithms (SHA-256 and SHA-256-PSK) are introduced.

[Figure 11-3](#) is a pictorial representation of the main blocks involved in PMF and how they interact.



**Figure 11-3 PMF Block Diagram**

hostapd is configured with PMF settings through the pmf.conf file. After successfully parsing pmf.conf, hostapd sends config ioctls down to WLAN driver. The Linux shim processes the config ioctls from hostapd and starts the AP with PMF settings in beacons. PMF STA associate with the AP and both AP and STA derive necessary keys. The key used for protecting unicast frames is used for protection of management frames as well.

### 11.5.2 Implementation

hostapd gets its PMF configuration parameters from pmf.conf file, including SSID, Authentication type, Encryption type, PMF flags (RSN caps) and Authentication Key Management (AKM) parameters. These configuration parameters are then sent down to the WLAN driver through ioctls. The WLAN driver (UMAC) creates a VAP with these parameters and starts beaconing with those settings.

### 11.5.3 Driver Files

- umac/mlme/ieee80211\_ie.c

The WLAN driver has been modified for RSN IE creation and parsing especially to handle new AKMs (SHA256 and SHA256-PSK) and RSN caps with PMF bits in both AP/STA modes.

- umac/mlme/ieee80211\_mgmt.c  
umac/mlme/ieee80211\_mgmt\_ap.c  
umac/mlme/ieee80211\_mlme.c

The above three files have functions to perform the following:

- To generate new MMIE for broadcast deauthorization/disassociation frames, the following files parse the association request and pass it to hostapd for proper association response.
- To set frame control protection bit for deauthorization/disassociation/action frame for the hardware to encrypt.
- To handle a SA\_QUERY request action management frame and send a SA\_QUERY response
- umac/crypto/ieee80211\_crypto\_pmf.c
  - To handle IGTK used for computing MIC in MMIE (Management MIC IE)
  - To figure out if PMF is enabled on a VAP and node
  - ieee80211\_cmac\_calc\_mic calculates MIC for the BIP\_CMAC\_128 and BIP-CMAC-256 modes
  - ieee80211\_gmac\_calc\_mic calculates MIC for the BIP\_GMAC\_128 and BIP-GMAC-256 modes
- umac/crypto/ieee80211\_crypto\_ccmp.c
  - ccmp\_encap allocates space for EIV & MIC for management frames when PMF is enabled
  - ccmp\_decap removes EIV & MIC bytes from plain management frames (the hardware decrypts but doesn't remove these bits)
  - Software MIC verification change for management frames if there is a big jump in packet numbers (PN)
- hal/ar9300\_reset.c (for AR9300 and newer chips)  
ar9300\_init\_mfp() enables the hardware to encrypt/decrypt management frames
- hal/ar5416\_reset.c (for AR5416 and newer chips)  
ar5416InitMFP() enables the hardware to encrypt/decrypt management frames

#### 11.5.4 hostapd Files

The following files are part of the Qualcomm Technologies private version distributed as part of the release package:

- athr-hostap/src/ap/drv\_callback.c

This file handles association comeback interval IE when association/re-association is rejected with status code 30 (association/reassoc rejected temporarily)

- athr-hostap/src/ap/ieee802\_11\_shared.c

The file has the SA\_QUERY request/response action frame generation, SA\_QUERY response handling.

#### 11.5.4.1 Sample hostapd configuration file (pmf.conf)

```
interface=ath0
bridge=br0
logger_syslog=-1
logger_syslog_level=2
logger_stdout=-1
logger_stdout_level=2
ctrl_interface=/var/run/hostapd
ctrl_interface_group=0
ssid=11wtest
dtim_period=2
max_num_sta=255
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wme_enabled=0
ieee8021x=0
eapol_version=2
eapol_key_index_workaround=0
eap_server=1
wpa=2
wpa_passphrase=12345678
wpa_key_management=WPA-PSK-SHA256
wpa_pairwise=CCMP
wpa_strict_rekey=1
ieee80211w=1
```

#### 11.5.4.2 hostapd usage

##### hostapd start command

```
hostapd -B 11w.conf
```

##### hostapd trace for sample configuration

```
/etc/ath # hostapd -d 11w.conf
random: Trying to read entropy from /dev/random
Configuration file: 11w.conf
ctrl_interface_group=0
eapol_version=2
Configure bridge br0 for EAPOL traffic. ieee80211_ioctl_siemode: imr.ifm_
active=393856, new mode=3, valid=1

br0: port 3(ath0) entering disabled state
atheros_set_privacy: enabled=0
atheros_receive_ DEVICE IS DOWN ifname=ath0
pkt Enter
SIOCG DEVICE IS DOWN ifname=ath0
IWRANGE: WE(compiled)=19 WE(source)=13 enc_capa=0xf
BSS count 1, BSSID mask 00:00:00:00:00:00 (0 bits)
```

```

Completing interface initialization
Flushing old station entries
atheros_sta_deauth: addr=ff:ff:ff:ff:ff:ff reason_code=3
atheros: set80211priv: ath0: ioctl op=0x8be6 (SETMLME) len=302 failed: 22
(Invalid argument)
atheros_sta_deauth: Failed to deauth STA (addr ff:ff:ff:ff:ff:ff reason
3)
Could not connect to kernel driver.
Deauthenticate all stations
atheros_sta_deauth: addr=ff:ff:ff:ff:ff:ff reason_code=2
atheros: set80211priv: ath0: ioctl op=0x8be6 (SETMLME) len=302 failed: 22
(Invalid argument)
atheros_sta_deauth: Failed to deauth STA (addr ff:ff:ff:ff:ff:ff reason
2)
atheros_set_privacy: enabled=0
atheros_del_key: addr=00:00:00:00:00:00 key_idx=0
atheros_del_key: addr=00:00:00:00:00:00 key_idx=1
atheros_del_key: addr=00:00:00:00:00:00 key_idx=2
atheros_del_key: addr=00:00:00:00:00:00 key_idx=3
atheros_del_key: addr=00:00:00:00:00:00 key_idx=4
atheros_del_key: addr=00:00:00:00:00:00 key_idx=5
Using interface ath0 with hwaddr 00:03:7f:12:30:e7 and ssid '11wtest'
Deriving WPA PSK based on passphrase
SSID - hexdump_ascii(len=7):
    31 31 77 74 65 73 74                                     11wtest
PSK (ASCII passphrase) - hexdump_ascii(len=8): [REMOVED]
PSK (from passph
    DES SSID SET=11wtest
rase) - hexdump(len=32): [REMOVED]
atheros_set_wps_ie buflen = 0
atheros_set_wps_ie buflen = 0
atheros_set_wps_ie buflen = 0
atheros_set_ieee8021x: enabled=1
atheros_configure_wpa: group key cipher=3
atheros_configure_br0: port 3(ath0) entering learning state
wpa: pairwise kebr0: topology change detected, propagating
y ciphers=0x8
abr0: port 3(ath0) entering forwarding state
theros_configure_wpa: key management algorithms=0x100
atheros_configure_wpa: rsn capabilities=0x0
atheros_configure_wpa: enable WPA=0x2
random: Cannot read from /dev/random: Resource temporarily unavailable
random: Got 0/20 bytes from /dev/random
random: Only 0/20 bytes of strong random data available from /dev/random
random: Not enough entropy pool available for secure operations
WPA: Not enough entropy in random pool for secure operations - update
keys later when the first station connects
GMK - hexdump(len=32): [REMOVED]
Key Counter - hexdump(len=32): [REMOVED]
WPA: group state machine entering state GTK_INIT (VLAN-ID 0)
GTK - hexdump(len=16): [REMOVED]
IGTK - hexdump(len=16): [REMOVED]
WPA: group state machine entering state SETKEYSDONE (VLAN-ID 0)
atheros_set_key: alg=3 addr=ff:ff:ff:ff:ff:ff key_idx=1
atheros_set_key: alg=4 addr=ff:ff:ff:ff:ff:ff key_idx=4
atheros_set_privacy: enabled=1

```

```

atheros_set_opt_ie buflen = 28
ath0: Setup of interface done.
12_packet_receive - recvfrom: Network is down
12_packet_receive - recvfrom: Network is down
Custom wireless event: 'Manage.auth 30'
atheros_raw_recv_11r: subtype 0xb len 30
    New STA
atheros_sta_auth: addr=00:24:d7:98:0f:68 reason_code=0
Custom wireless event: 'Manage.assoc_req 99'
atheros_raw_recv_11r: subtype 0x0 len 99
ath0: STA 00:24:d7:98:0f:68 IEEE 802.11: associated
STA did not include WPS/RSN/WPA IE in (Re)AssocReq
No WPA/RSN IE from STA

```

## 11.5.5 Data Receive Aggregation

### 11.5.5.1 A-MPDU Receive Processing

When an ADDBA request is received, resources are allocated to order packets by sequence number as they are received (*ath\_addba\_requestprocess*). If these resources cannot be allocated, a negative acknowledgment is transmitted to the sender. When these resources have been successfully set up, the next expected sequence number is obtained from the ADDBA request.

As A-MPDUs are received for the TID, they are processed by *ath\_ampdu\_input()*. The TID and sequence number is extracted from the wireless packet header, and the receive reorder queue corresponding to this TID/node is retrieved. If the sequence number of the packet is past the current window, then all packets in the receive window are indicated to the protocol. The new packet received is added to the window if there is not already a packet with the same sequence number. After putting the packet into the window, the window is checked, and if there are in-order packets that can be indicated to the protocol, these are pulled out and handed up. If there is a hole in the window, the indication stops at that point. The window is then moved by the number of packets that were removed for indication.

### 11.5.5.2 Timeout for Holes

Packets are not always received in sequence due to the nature of the wireless medium. Missing packets are termed “holes”. Packets can be indicated to the protocol stack only in sequence. However, there could be conditions under which a missing packet is never received. Since the hardware cannot wait indefinitely for such holes to be filled, a time out period is started, and when the time expires, all packets that were pending in the window are indicated to the protocol (*ath\_rx\_timer*) At this point, the next expected sequence number is pushed past the sequence number of the last indicated packet.

## 11.5.6 A-MSDU Aggregation

A-MSDU aggregation is another way of aggregating packets. The sender packages multiple Ethernet packets into a single wireless packet and queues it for transmission. When the receiver gets this packet, each component packet is extracted and handed up to the protocol stack.

A-MSDU aggregate reception is mandated by the 802.11n standard for the normal ACK policy and is negotiated when using A-MPDU (Block ACK). Transmission of A-MSDUs is optional. The presence of an A-MSDU aggregate is indicated by the AMSDU bit in the QoS header. A-MSDU size is restricted to a maximum of either 3839 or 7935 bytes, and the value to be used is provided in the HT-Capabilities IE. The Qualcomm driver restricts it to 3839 bytes for receive. Note that A-MSDUs can be aggregated into A-MPDUs.

Each packet (sub-frame) that is aggregated into an A-MSDU has the following format: DA, SA, and length, followed by an LLC header. Each sub-frame is rounded up to be a multiple of four bytes (with a padding of 0-3 bytes). Note that any padding is not reflected in the packet length field in the sub-frame header.

As specified earlier, transmitting an A-MSDU is optional. The reason for transmitting A-MSDUs is mainly that A-MSDUs achieve greater medium efficiency.

The Qualcomm driver has code to aggregate TCP ACKs. TCP ACK packets are typically small packets (58 bytes in Ethernet form). They could be sent within an A-MPDU if a Block ACK has been negotiated. However, if they are sent as a normal A-MPDU, the per-packet overhead in terms of MPDU delimiters and wireless headers is significant in relation to the size of the packet. However, if TCP-ACKs are sent as A-MSDUs (which potentially, can be further aggregated into A-MPDUs), the overhead is removed and a net bandwidth improvement is achieved. Internal measurements showed as much as an 8-10% increase in TCP throughput when A-MSDUs are used.

## 11.6 Configure AMSDU per VAP for DA and OL modes

This section describes the support for Aggregated-MAC Service Data Unit (AMSDU) per VAP for both direct attach (DA) and offload (OL) modes. Until QCA\_Networking\_2016.SPF.4.0 release, AMSDU support was available on the athX interfaces (protocol layer) in OL mode only. Also, this support was implemented per radio. Starting with QCA\_Networking\_2017.SPF.5.0, the following enhancements are made:

- AMSDU support is per VAP and not per radio
- AMSDU support is extended for DA mode

Because it is necessary in certain customer deployments to lower AMPDU and AMSDU to avoid frame drop to certain VAPs and not cause the throughput drop with other VAPs, this functionality is made compatible and consistent.

Specific commands are available for AMSDU settings to enable/disable AMSDU and set the number of sub-frames limit.

In DA mode, enter the iwpriv athX amsdu <0/1> command to enable/disable AMSDU per VAP. By issuing the command, the flag IEEE80211\_FEXT\_AMSDU for IC is set or cleared.

### AMSDU command guidelines

- In DA mode, enter the iwpriv wifiX amsdu <0/1> command, which is an equivalent command of the iwpriv athX command on wifiX interface to disable/enable AMSDU. This command calls ath\_set\_config with parameter ATH\_PARAM\_AMSDU to set sc to sc\_txamsdu.

- In DA mode, enter the iwpriv wifiX amsdu {1-31} command to set AMSDU maximum frames limit per radio. A known limitation exists with the DA implementation. It only supports small packet AMSDU, if the packet length is greater than 100, aggregation to AMSDU is not performed.
- In OL mode, enter the iwpriv athX amsdu {1-31} to set AMSDU subframes limit to target.
- In both DA and OL modes, for AMSDU support per VAP, an effective approach is to use “iwpriv athX amsdu {0-31}” to enable/disable AMSDU and set AMSDU subframes limit. The value 0 indicates that AMSDU is disabled and the values 1-31 indicate the AMSDU subframes limit. The radio-level AMSDU setting for DA path that is available using the iwpriv wifiX amsdu <0 |1> command remains unchanged. The VAP-level setting and radio-level setting are independent of each other. Only when both are enabled in DA mode, AMSDU is activated.
- Enter the iwpriv athX get\_amsdu command to retrieve the AMSDU parameters per VAP.

## 11.7 Using SA query timeout for PMF

The 802.11w standard called Protected Management Frames (PMF) that shields the client by using a Security Association (SA) teardown protection mechanism. PMF requires the access point (AP) to check with the legitimate client first by sending a Security Association (SA) Query Request frame to the legitimate client. The legitimate 802.11w client must respond with a Security Association (SA) Query Response frame within a predefined amount of time (milliseconds) called the SA Query Retry time. If the legitimate client responds in time, then legitimate client maintains the connection and the AP sends the unauthorized client a status code 30 message that denotes that the association request be rejected temporarily and to be attempted later. This action will prevent the unauthorized client from connecting and prevent the legitimate client from being disconnected from the AP.

Starting with QCA\_Networking\_2017.SPF.5.0.3, when an AP receives an association or reassociation request from an associated client which is in forwarding state, an association response with the status code of 30, including the Timeout interval IE, must be sent by the driver. The SA Query procedure will be started by hostapd after the event is received from the driver, including the timeout interval IE.

Currently, hostapd handles the association response and sends association response with status code 30 including the Timeout interval IE and starts SA Query procedure according to the timeout interval IE. However, if the legitimate client does not reply in time (milliseconds) to the SA Request frame, then the client session is torn down by the AP by sending a disassociation message.

When an associated PMF client sends an association/reassociation request, the association response including the timeout interval IE from hostapd is dropped in the driver. The driver sends an association response with IEEE80211\_STATUS\_REJECT\_TEMP, status code 30, and timeout interval IE of type “Association comeback time to the hostapd/application.”

Based on this timeout interval IE, the driver sends an event to hostapd/ application and hostapd starts the SA Query procedure after receiving the association response event from the driver. Based on the timeout interval IE, association comeback occurs after SA Query procedure is completed. The comeback time before which the STA must associate with the AP depends on the timeout IE value.

Enter the iwpriv athX timeoutie “value” command to specify the timeout interval after which hostapd starts the Security Association (SA) Query procedure upon receiving the association response event from the driver. To display the configured SA query timeout interval, enter the iwpriv athX g\_timeoutie command.

To enable PMF using the timeout interval mechanism for Security Association (SA) query, enter the iwpriv athX pmf\_assoc 1 command. To check whether PMF using SA query timeout interval is configured, enter the iwpriv athX g\_pmf\_assoc command. Alternatively, enter the following UCI commands to enable this feature:

```
uci set wireless.@wifi-iface[X].pmf_assoc=1
uci commit wireless
```

Both the AP and STA can be enabled with PMF by using uci set wireless.@wifi-iface[X].ieee80211w=2 command.

For a non PMF client, or if “pmf\_assoc” is not enabled the association response should not be dropped from hostapd nor should the driver send an association response frame.

### 11.7.1 Protected management frames (PMF) parameters

**Table 11-8 Protected management frames (PMF) parameters**

| Parameter                           | Command   | Description  |
|-------------------------------------|---|--|
| timeoutie<br>“value”<br>g_timeoutie | iwpriv athX timeoutie<br>“value”<br>iwpriv athX g_timeoutie | Enter the iwpriv athX timeoutie “value” command to specify the timeout interval after which hostapd starts the Security Association (SA) Query procedure upon receiving the association response event from the driver.<br>To display the configured SA query timeout interval, enter the iwpriv athX g_timeoutie command. |
| pmf_assoc <1   0><br>g_pmf_assoc    | iwpriv athX pmf_assoc <1   0><br>iwpriv athX g_pmf_assoc    | Enable (1) or disable (0) PMF using the timeout interval mechanism for Security Association (SA) query.<br>To check whether PMF using SA query timeout interval is configured, enter the iwpriv athX g_pmf_assoc command. Alternatively, enter the following UCI commands to enable this feature:                          |

### 11.7.2 UCI command to configure SA query timeout for PMF

To enable PMF using the timeout interval mechanism for Security Association (SA) query, enter the iwpriv athX pmf\_assoc 1 command. To check whether PMF using SA query timeout interval is configured, enter the iwpriv athX g\_pmf\_assoc command. Alternatively, enter the following UCI commands to enable this feature:

```
uci set wireless.@wifi-iface[X].pmf_assoc=1
uci commit wireless
```

Both the AP and STA can be enabled with PMF by using uci set wireless.@wifi-iface[X].ieee80211w=2 command.

For a non PMF client, or if “pmf\_assoc” is not enabled the association response should not be dropped from hostapd nor should the driver send an association response frame.

## 11.8 Retrieve and display radar channel details

The functionality to obtain radar channel information and display the collected details is implemented. This section describes the design for fetching radar channel information and notification.

In current mode, users can retrieve the channels that detected radar. If no channel is available in high band (5470~5725) in current mode, the user is notified. Similarly, if a channel is available in high band, the user is notified.

To get channel radar information in current mode, the wlanconfig athX list channel command is used, and the radar information in current mode is displayed. The collected detail is sent to user space in driver.

The example code as follows:

```
int wlan_get_chaninfo(wlan_if_t vaphandle, int flag, struct ieee80211_
channel *chan, int *nchan)
{
.....
    Clear all the radar information
    For all the channels, return to host, get its channel structure based on
    current mode.
    If the channel is marked as radar, get the extension channel, and mark
    all the primary channel and extension channels as radar
}
```

In wlanconfig user space, if one channel is marked as IEEE80211\_CHAN\_RADAR, the channel is marked as “#”. In this manner, wlanconfig is used to list the radar info in current mode. The wlanconfig change currently is only for test purposes because users apply their own IOCTL to get the RADAR info.

To notify the channel radar information in user space, in ol\_if\_dfs\_clist\_update function, when NOL is added or removed, this function is called to update the radar information of all the channels. In this function, a check is performed in current mode to determine if all the channels are marked with radar in high band. If so, notify this info to user space. If a channel that is not marked as radar is present in high band, this information is also notified to user space.

To notify user space about the event, the WIRELESS\_SEND\_EVENT API with IWEVCUSTOM is used to send the event to user space.

### 11.8.1 Sample configuration scenario

1. Set AP as 11ACVHT80 mode, set in radar channel, and trigger the radar in this channel. The output of the wlanconfig ath0 list channel command displays all the 80M bandwidth channels that are marked as RADAR.

2. Set AP as 11ACVHT40 mode, set in radar channel, trigger the radar in this channel. The output of the wlanconfig ath0 list channel command displays all the 40M bandwidth channel are marked as RADAR.
3. After 30 minutes, the radar channel expires. The output of the wlanconfig ath0 list channel command displays all the 80/40M channel are not marked as RADAR.
4. Trigger the radar for all the channels in (5740~5725) A printk (message send to host) displays that no channel is available.
5. After 30 minutes, high band channel radar expires. A printk (message send to host) displays the channels that are available.

QUALCOMM®  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

# 12 QoS and Policy Control Operations

---

This chapter describes the quality of service (QoS) capabilities, such as access control lists (ACLs) and iQue (Intelligent QoS for User Experience), to specify policies and filters for controlling and classifying the manner in which traffic must be forwarded for processing.

## 12.1 Access Control List

Access Control List (ACL) is a mechanism in AP mode operation to restrict association of clients based on their MAC addresses. User can add MAC addresses to the ACL and also configure policy to be used for allowing client association based on this list. The policy can be either to allow or deny association of clients with MAC address matching one of the addresses in the ACL.

### 12.1.1 Implementation

A hash list of addresses is maintained by software and an API is provided to add and remove addresses to this list. An API is also provided to set the ACL policy. The ACL policy is set to OPEN by default, which means no check is done against ACL when clients associate with AP.

The ACL feature is implemented in *umac/acl/ieee80211\_acl.c*. There is also a secondary ACL list implemented for band-steering purpose.

The primary and the secondary ACL list share the same list and are differentiated by flags.(i.e) IEEE80211\_ACL\_FLAG\_ACL\_LIST\_1 denotes primary list and IEEE80211\_ACL\_FLAG\_ACL\_LIST\_2 denotes secondary ACL list. The access control list is created and initialized by *ieee80211\_acl\_attach* which is invoked during AP initialization (*ieee80211\_vap\_setup*).

The following functions take the flag argument that specifies the ACL list (primary or secondary) that does the corresponding function on the specified ACL list. *ieee80211\_acl\_add* adds a given MAC address to the hash list and *ieee80211\_acl\_remove* removes the given address from the specified ACL list. *ieee80211\_acl\_setpolicy* sets the ACL policy to allow or deny clients based on the list. The address check is done by the function *ieee80211\_acl\_check* which is invoked whenever an authentication request is received from the client (from function *mlme\_recv\_auth\_ap*). The hash list is searched for the MAC address of the client and authentication is allowed or denied based on the ACL policy configured. The following APIs are provided to manage ACL:

*wlan\_set\_acl\_policy* is used to set the ACL policy to one of the following:

- IEEE80211\_MACCMD\_POLICY\_OPEN: Allow all clients without any checking
- IEEE80211\_MACCMD\_POLICY\_ALLOW: Allow client only if its MAC address is present in the given ACL

- IEEE80211\_MACCMD\_POLICY\_DENY: Allow client only if its MAC address is not present in the given ACL
- IEEE80211\_MACCMD\_FLUSH: Flush (remove) all the addresses from the given ACL
- IEEE80211\_MACCMD\_DETACH: Detach the ACL module and free all related data structures in the given ACL.

*wlan\_get\_acl\_policy*: This retrieves the current ACL policy configured.

*wlan\_set\_acl\_add*: This adds a given MAC address to the specified ACL.

*wlan\_set\_acl\_remove*: This removes a given MAC address from specified ACL.

*wlan\_get\_acl\_list*: This retrieves the addresses present in the specified ACL.

## 12.1.2 Configuration

The configuration to support association denial notification can be enabled/disabled and queried with below iwpriv commands:

```
iwpriv athX acl_notify 1|0
iwpriv athX get_acl_notify.
```

The following commands set the ACL policy to deny association of clients with addresses matching one the entries in ACL and to add an address to the ACL.

To add to the primary ACL, use the following commands:

```
iwpriv ath0 maccmd 2
iwpriv ath0 addmac 00:03:7f:00:0a:0d:0d
```

To add to the secondary ACL, use the following commands:

```
iwpriv ath0 maccmd_sec 2
iwpriv ath0 addmac_sec 00:03:7f:00:0a:0d:0d
```

## 12.2 iQue (Intelligent QoS for User Experience)

Intelligent QoS for User Experience is a group of features that are designed to provide differentiated quality of service to different traffic types. It currently consists of two features

- Head of the Line Blocking Removal (HBR)
- Multicast Enhancement (ME)

### 12.2.1 Multicast Enhancement (ME)

#### 12.2.1.1 Theory of Operation

Multicast Enhancement is a feature that removes the inefficiency of multicast transmission by converting a single multicast frame into multiple unicast frames.

802.11 WLAN does not specify a method for link layer multicast traffic to be reliably transmitted using the link layer acknowledgment protocol. To compensate for this, most implementations use the lowest transmission rate to transmit link layer multicast traffic to ensure that the frame is received correctly by all STAs. This is an inefficient use of the capacity of the medium if there is just one or two STAs registered to receive the multicast, especially if the link budget to the STAs can support high transmit rates.

Multicast enhancement overcomes this inefficiency. A single multicast frame is converted to multiple unicast frames and they are transmitted to the STAs that have registered (joined the multicast group) to receive the multicast frame. The transmission rate that is used is the best transmission rate that the STA and the link budget can support. The feature is intended to be used for small home networks where the number of STAs is small (< 10) and the number of multicast groups is small (< 10). It is primarily meant for IPTV deployments where video traffic is multicast.

### 12.2.1.2 Implementation

The AP inspects all incoming traffic (both traffic received on the WLAN and traffic to be sent on the WLAN).

The AP searches for IGMP JOIN frames, and detects when one STA tries to join a specific multicast group. The AP maintains a table of the MAC addresses of the STAs and the corresponding group addresses of the multicast groups to whom each STA belongs. On detection of a STA that is joining a multicast group, it then adds the MAC address of the STA and the group address into the table, if they are not present in it.

The AP also searches for IGMP LEAVE packets and deletes the corresponding entry in the table in case a STA leaves the group (for example, the user closes the IPTV application). The AP may delete an entry in the table if there is no traffic for the group during an arbitrary time window. For example, if the user pauses the IPTV application long enough or the multicast service terminates without notification.

The AP also searches for multicast frames. On reception of a multicast frame whose group address is present in the conversion table, the AP converts this frame into several unicast frames, one for each STA that is a member of the group address in the table, and sends each frame to the destination.

There are two operating modes:

- **Tunneling Mode**

The AP encapsulates the multicast 802.3 frame by taking the frame as the payload of several new 802.3 frames. For the i-th new frame, a header with the MAC address of the i-th associated STA as the destination address is added. This option requires the STAs to support the tunneling mode.

- **Translating Mode**

The AP changes the multicast address of the 802.3 frame into the MAC addresses of each associated STAs. This option does not need any support on the STAs. However if the application expects the link layer frame to have a multicast address, the application may fail.

### 12.2.1.3 Code

The code for ME is in the folder wdrivers/wlan/umac/ique. The files for ME are ieee80211\_me.c, ieee80211\_me\_priv.h

### 12.2.1.4 Configuration

The following are the commands used to configure Multicast Enhancement. The commands should be executed separately for each VAP.

- `iwpriv athX mcastenhance <param>`  
`iwpriv athX get_mcastenhance`  
 Set/get mcast enhancement mode.  
 Mode 0: Disable multicast enhancement.  
 Mode 1: Tunneling mode.  
 Mode 2: Translating mode.
- `iwpriv athX medump`  
 Dump the snoop table for multicast enhancement.
- `iwpriv athX medebug <param>`  
`iwpriv athX get_medebug`  
 Set/get debug level for multicast enhancement.
- `iwpriv athX me_length <param>`  
`iwpriv athX get_me_length`  
 Set/get length of the snoop table.
- `iwpriv athX metimeout <param>`  
`iwpriv athX get_metimeout`  
 Set/get the timeout for a STA to be removed from the snoop table if it is idle.

### 12.2.1.5 Design Considerations for Offload Architecture

When the IGMP Snooping and Multi-cast Enhancement features are enabled, Multi-cast frames can be identified based on the following:

- Host inspection for multi-cast frames  
 Host inspects all the outgoing frames for multi-cast type and performs multi-cast to unicast conversion as needed.
  - Pros: Simple design.
  - Cons:  
 Every frame needs to be checked irrespective of type. This check is already performed in target and this is just a duplicate check on host.
- Firmware inspection for multi-cast frames  
 Host does not do any check. Firmware as part of its current check, flags multicast frames as TX\_HOST\_INSPECT and returns it to host for further processing.

- Pros: No additional check in data path for regular traffic
- Cons
  - Adds some latency to multicast frames due to host inspecting frame after target determines it as a multicast frame.
  - Drastically increases the latency, bus transactions, power in case video related test cases.
  - Extra descriptors are wasted since it gets consumed in host inspect path.

Host inspection for multi-cast frames is used by default for this implementation. For better handling of TSO, scatter gather and multi-cast enhancement and to minimize the performance during normal data transfer, data path is planned to have feature and performance path. TSO, scatter gather and multi-cast enhancements will add one check at the beginning of the data path to choose which path the packet needs to take. TSO, scatter gather and multi-cast enhance will use feature path.

### Pseudo code

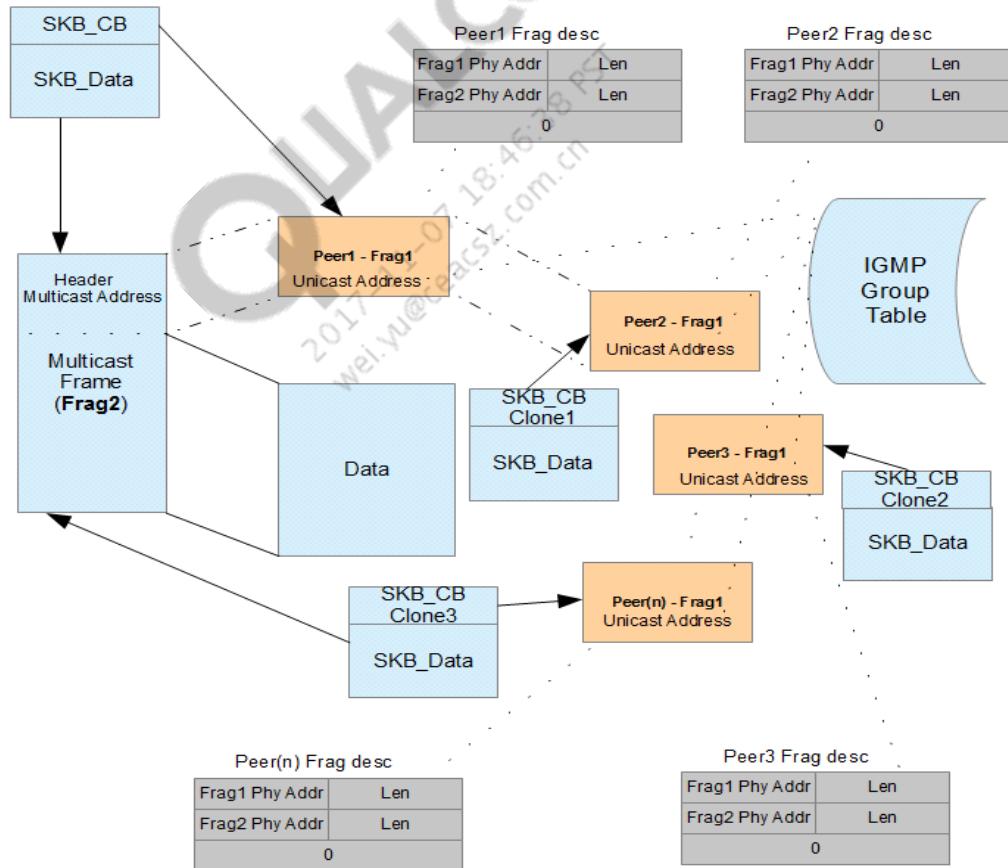
```
// Pseudo code for HARD_START OS/other subsystem call handler
HARD_START handler
For each frame indicated in list DO
  If Multicast enhancement feature is enabled AND multicast frame Then Call
    mcast to ucast conversion routine.
  If mcast to ucast routine indicates that packet is handled Then
    Return. /*No more handling required */
  Else
    /* Existing logic takes care of transmitting multicast frame.*/
  End If
End If
DONE
End For
```

Once host determines that the packet is multi-cast, it does further processing to convert it to unicast.

1. Existing API "ieee80211\_me\_SnoopListGetMember" is used to get the list of members belonging to a group.
2. To avoid creating multiple copies of whole packet, multi-cast packet is split into two fragments.
  - Fragment-1 contains, header + additional data required by FW. (Using API "htt\_pkt\_dl\_len\_get()" to get this information)
  - Fragment-2 contains the rest of data.
3. For each group member, one copy of Fragment-1 is created. Destination MAC address in this fragment is changed to macaddress of the member and Fragment-2's usage count is incremented.
4. Fragment-1 for each peer is chained in a single linked list so that it can be freed once all the TX is complete.
5. ol\_tx\_ll\_fast (feature path) is called to transmit frame. SKB's CB is used to maintain a type and pointer type will be set as MC\_UC and pointer will point to the frag1 of current tx.
6. If the type is MC\_UC, fragment descriptor is programmed as follows:

- Frag1 physical address = Physical address of Fragment-1 pointed in SKB's CB pointer.
  - Frag1 len = Len of Fragment-1.
  - Frag2 physical address = Physical address of SKB->data + Len of Fragment-1
  - Frag2 len = SKB->len - Len of Fragment-1
7. If the length of frame to be transmitted is less than "htt\_pkt\_dl\_len\_get()", then only one framgement is programmed (Fragment-1).
8. FW transmits frame as though it is normal unicast frame. (No change required here)
9. On TX\_COMPLETE. Host decrements the usage count of SKB. Once it is determined that it is not shared, Fragment-1 for all peer is freed by traversing the SKB's CB pointer. And the complete SKB is unmapped and freed once the reference count reaches 1 (i.e., not shared). Using skb clone takes care of reference count implicitly.

Figure 12-1 depicts the data after conversion.



Example: Depicting conversion for 4 clients.

Figure 12-1 Example depicting conversion for four clients

## Debug Stats

IQUE support provides the following debug stats

- iwpriv athX medump - Display snooping table
- iwpriv athX me\_showdeny - Display deny table

The following will be added for the present release:

- iwpriv athX txrx\_fw\_stats 12 - Display mcast stats
- iwpriv athX txrx\_fw\_st\_rst - Clear mcast stats.

Following debug stats will be maintained under compile time flag and these will be displayed through iwpriv commands. (There will be iwpriv command to reset these stats)

1. mcast\_to\_icast count - Number of mcast frames translated.
2. mcast\_to\_no\_conversion - Number of mcast frames transmitted without conversion. (when mcast enhance is not enabled)
3. mcast\_dropped\_no\_member - Number of mcast frames dropped since there were no members in the list.
4. mcast\_dropped\_internal\_error - Number of mcast frames dropped due to internal errors like lack resources (tx\_desc or netbuf)

## 12.2.2 Head of the Line Blocking Removal (HBR)

### 12.2.2.1 Theory of Operation

Head of the Line Blocking is a feature that prevents the traffic to one STA from occupying a large portion of the air time if the STA is non-responsive.

In a WLAN network, a STA may become non-responsive if it moves out of range, or scans on another channel, or enters power save mode, or resets. In some of these cases it may not inform the AP that it is entering a non-responsive state. If a UDP stream is being sent to that STA, the link layer frames will not be acknowledged at the link layer, and will be retransmitted several times at progressively lower transmission rates. UDP does not implement a reliability mechanism (such as ACKs) and has no method to pause transmission if one of the end points is non-responsive. This condition will persist indefinitely until some other mechanism corrects this error condition. The UDP frames, the ACKs and the retransmissions will occupy a significant fraction of the medium's capacity.

Video along with other types of traffic is often uses UDP as the transport protocol. HBR on the AP deals with this problem by detecting this condition and pausing transmission of frames to a STA in a non-responsive condition.

### 12.2.2.2 Implementation

The implementation uses a Finite State Machine (FSM) for each associated STA in each VAP. The state machine is called `ieee80211_hbr_list` for each VAP, with its entries (state machines) called `ieee80211_hbr_sm`, one for each STA.

There are three states for the state machine: *ACTIVE*, *BLOCKING*, and *PROBING*. By default, all STAs are in *ACTIVE* state.

The AP monitors the traffic in the video queue for all STAs. This is implemented through the rate control module during the rate control find operation (pre-transmit) and rate control update operation (post-transmit). If the video stream to a STA is transmitted at the lowest allowable rate with its PER higher than the PER-high-threshold, the channel condition to this STA is not reliable and the STA is considered to be in a non-responsive state. In this condition, traffic to this STA will occupy a significant fraction of the medium capacity, and it may block the video streams to other STAs.

The AP blocks this video stream temporarily by switching the state of the corresponding state machine from *ACTIVE* to *BLOCKING* and then to *PROBING*. In both the *BLOCKING* and *PROBING* states, the streams in the VI queue to this specific STA are dropped. The AP tests the channel condition if the state machine is in *PROBING* state by sending out QoS Null frames to the blocked STA at the lowest allowable rate. If the rate control module detects that the PER to the blocked STA is lower than the PER-low-threshold, it indicates that the channel condition to the blocked STA has been recovered and the state machine of the blocked STA switches back to the *ACTIVE* state and unblock all traffic to this STA.

#### 12.2.2.3 Code

The code for HBR is in the folder wdrivers/wlan/umac/ique. The files for HBR are ieee80211\_hbr.c, ieee80211\_hbr\_priv.h

### 12.3 Support for SFE on IPQ8064 platforms—Coexistence between offload engines

This section describes the three types of coexistence about acceleration engines. These three types of coexistences are as follows:

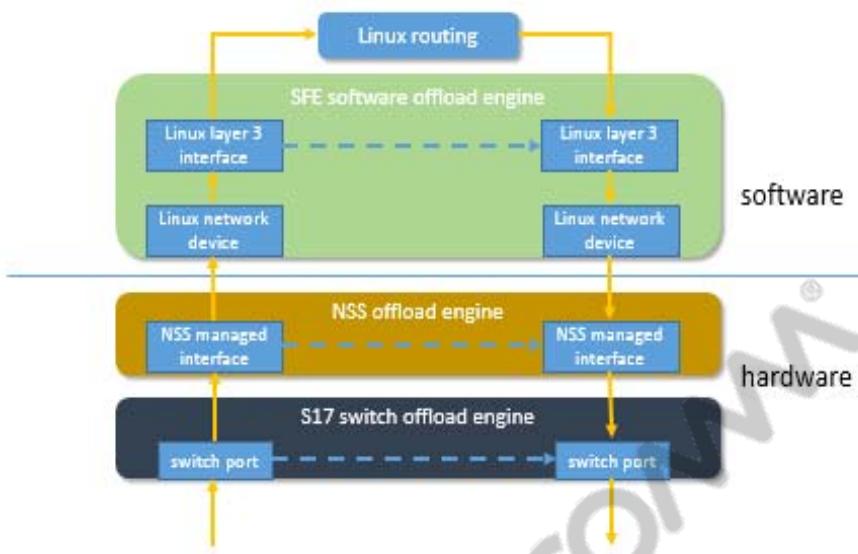
- Coexistence of offload engines
- Coexistence of connection manager
- Coexistence of front-end in ECM

#### Coexistence of offload engines

The following three types of offload engines are present:

- NSS offload engine, which is a hardware engine in IPQ8064 platform.
- Switch HNAT offload engine, which is a hardware engine in S17 switch.

Shortcut forward engine (SFE) is an offload engine; it is a pure software engine run as a Linux kernel module.



The existence of NSS offload engine or S17 switch offload engine depend on the board type. The SFE offload engine is a pure software that does not depend on any hardware, it can run in any Linux operate system. Normally offload engine will not take effect if no connection manager enable it. Actually SFE offload engine is always loaded when system starts. There is no problem even all kinds of offload engine coexist in the board because they don't take effect before connection manager enable them.

### Coexistence of connection manager

Connection manager is the control plane of acceleration. A single connection manager can manage any number of offload engines. Therefore, to avoid an offload engine from being managed by two connection managers simultaneously, multiple connection managers must not be running concurrently.

The following three types of connection managers exist:

- ECM is for both NSS offload engine and SFE software offload engine.
- SFE connection manager is for SFE software offload engine only.
- HNAT connection manager is for S17 HNAT offload engine only.

The ultimate goal is to merge the aforementioned three connection managers into one connection manager. Also, ECM is the destination place. However, currently, it is necessary to allow the coexistence of multiple connection managers.

The highest priority is assigned to ECM, and the lowest priority is assigned to HNAT connection manager. Other two connection managers must not become effective when ECM exists. Also, HNAT connection managers do not become effective when the SFE connection manager exists.

Connection managers with lower priority detect the existence of connection managers with higher priority. Every connection manager contains a methodology to show their existence. The following are the methodologies using which each connection manager shows its existence:

- ECM is identified to exist and run on the system if the “/sys/kernel/debug/ecm” directory is present. It is not necessary to detect the connection manager with higher priority in this case because ECM has the highest priority.
- SFE connection manager is identified to exist and run on the system if the “/sys/module/shortcut\_fe\_cm” file is present. To detect the connection manager with higher priority, the “/sys/kernel/debug/ecm” directory is detected in the “/etc/init.d/shortcut-fe” initial script. This initial script does not load the SFE connection manager kernel module if ECM is detected.
- A methodology to identify the existence of HNAT connection manager is not currently available. Similarly, an approach to detect the connection manager with higher priority with HNAT running is not currently available.

## Coexistence of front-end in ECM

There are two front ends in ECM. NSS front end accelerate flows in NSS hardware offload engine. SFE front end accelerate flows in SFE software offload engine. In theory they can coexist at the same time. But there are some potential risks indeed, such as conflict about statistics sync. To avoid such potential risks, we decide to run only one front end at runtime.

A kernel module parameter “front\_end\_selection” is used to decide which front end must run. This parameter has 3 possible values:

- ‘0’: Automatic selection. In this mode, ECM will detect board type to see if NSS acceleration engine is supported. If yes, NSS front end will be selected to run. Otherwise SFE front end will be selected to run.
- ‘1’: NSS front end. In this mode, ECM always select NSS front end to run except board don’t support NSS acceleration engine.
- ‘2’: SFE front end. In this mode, ECM always select SFE front end to run.

Users can enter the “insmod ecm” command to run ECM in automatic selection mode. Similarly, enter the “insmod ecm front\_end\_selection=1” command to select NSS front end for ECM, and the “insmod ecm front\_end\_selection=2” command to select SFE front end for ECM.

The board has a device tree that signifies the board type. ECM uses the following code to select front-end in its initial function:

```
int ecm_front_end_ipv4_init(int front_end_selection){
    int nss_supported = of_machine_is_compatible("qcom,ipq8064") ||
                       of_machine_is_compatible("qcom,ipq8064");

    if ((front_end_selection == 0 || front_end_selection == 1) && nss_supported) {
        return ecm_nss_ipv4_init();
    }
    if (front_end_selection == 0 || front_end_selection == 2) {
        return ecm_sfe_ipv4_init();
    }
    return -1;
}
```

After a front-end is selected, a file is exported to identify its existence.

- For NSS front-end, the “/sys/kernel/debug/ecm//ecm\_nss\_ipv4/stop” file is exported.
- For SFE front-end, the “/sys/kernel/debug/ecm/ecm\_sfe\_ipv4/stop” file is exported.

## 12.4 Reject clients with low SNR

The primary focus of this implementation is to maintain the clients with reasonably-high signal to noise ratio (SNR) values in BSS. Low SNR based clients are deauthenticated and not allowed in BSS until SNR improves for such clients. Implementation of this functionality is independent of architecture (that is, this mechanism works with direct attach and partial offload modes).

The following commands are added to configure SNR-based client rejection:

1. iwpriv wifiN set\_min\_snr\_en <0/1>: 1 – Enables SNR accounting, 0 – Disables SNR accounting
2. iwpriv wifiN get\_min\_snr\_en: Returns current status of SNR accounting
3. iwpriv wifiN set\_min\_snr <SNR threshold in dB>: Sets SNR threshold value
4. iwpriv wifiN get\_min\_snr: Retrieves current value for SNR threshold

### Direct attach

A check for SNR values occurs in receive path whenever SNR of a particular client falls below a preconfigured value. In such a condition, deauthentication frames are sent to the client. The client is not allowed to connect to the BSS until the SNR for the client exceeds the configured threshold.

### Offload

Peer statistics (periodic events from firmware with configured interval in seconds) are used as the criterion to perform SNR verification. Whenever the peer statistics indicate that SNR is deteriorating, deauthentication frames are sent to the client and the client is not allowed to connect until SNR improves.

SNR accounting is active for all frames (including management) and is based on each radio. Therefore, enabling and disabling SNR accounting impacts all the VAPs in a particular radio.

**NOTE** With QCA9880 firmware, when several nodes are connected, the firmware does not always send the statistics for all peers due to less memory. This condition causes a delay in retrieving the statistics for a particular node. The statistics message from the firmware is transmitted at a periodic interval. The host learns the SNR of the STA only in this message. As a result, as soon as the message is processed, the deauthentication frames are sent out by the host. This delay is an expected behavior with SNR-based client rejection enabled in a scenario consisting of a single client and a single VAP.

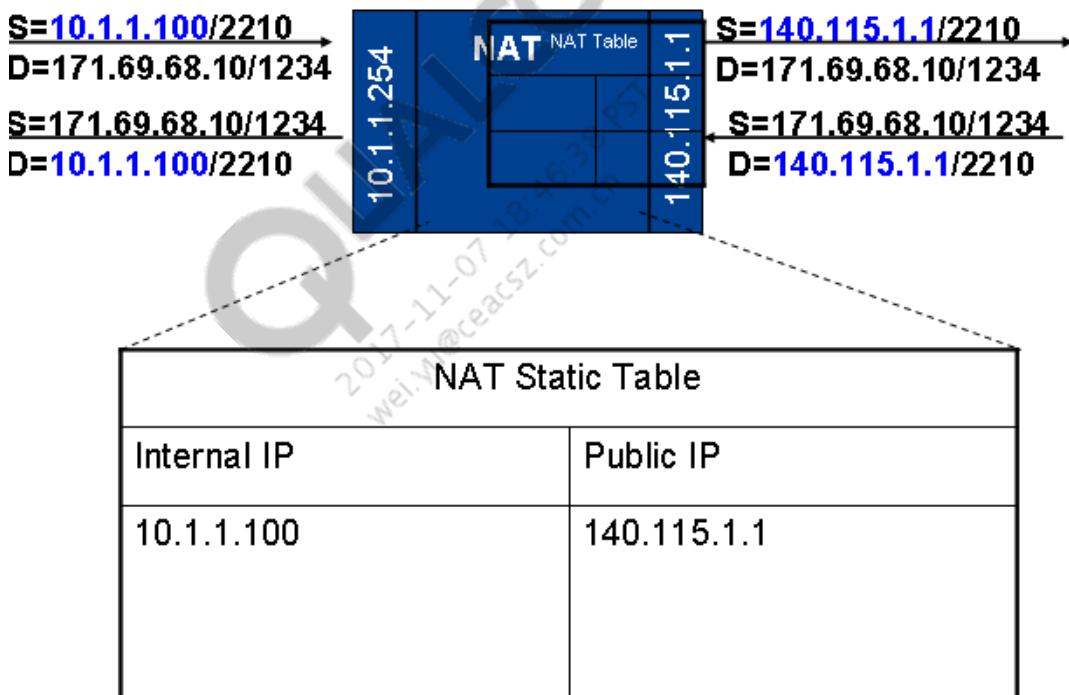
## 12.5 HNAT on QCA9531, QCA9558, and QCA9563 chipsets

For host network address translation (HNAT), both static NAT and network address port translation (NAPT) functions are supported. The HNAT module is built into SSDK kernel module. When SSDK module is loaded, the HNAT function is disabled by default.

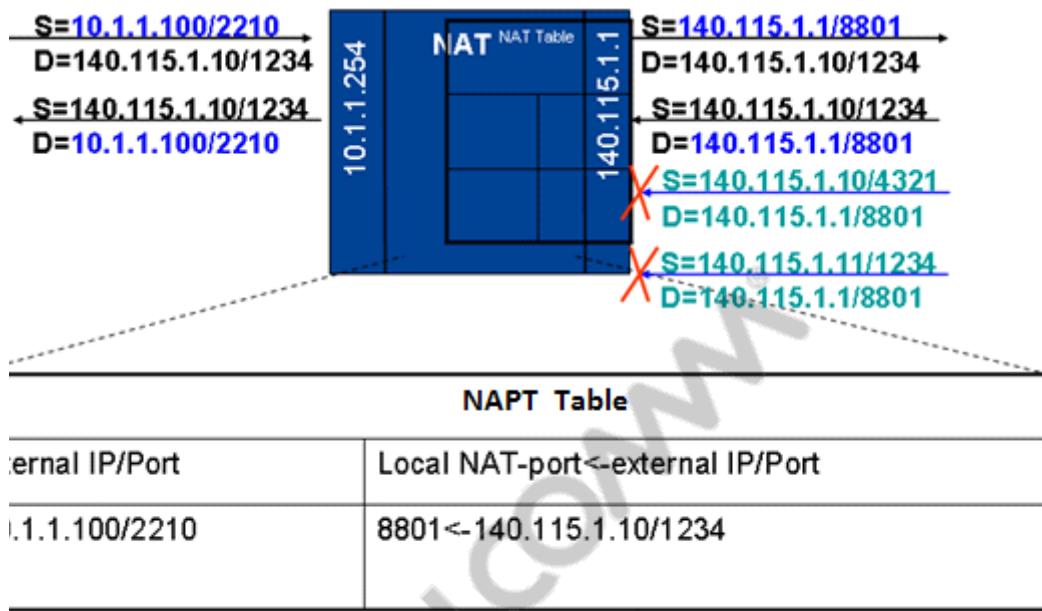
This section describes the programming details for the hardware network address translation (NAT) engine inside the QCA9531, QCA9558, and QCA9563 chipsets. For QCA9531, QCA9558, and QCA9563 chipsets hardware NAT engine, the following specifications apply:

- 16 PPPoE session tables
- 128 ARP entries
- 1024 NAT entries
- 8 router MAC address/VID table
- 16 public IP tables
- 8 NAPT counters

Both static NAT and NAPT functions are supported. Static NAT uses a simple one-to-one mapping of private and public addresses, and functions as shown in the following illustration:



For NAPT, the port restricted mode is used, and it functions as shown in the following figure:

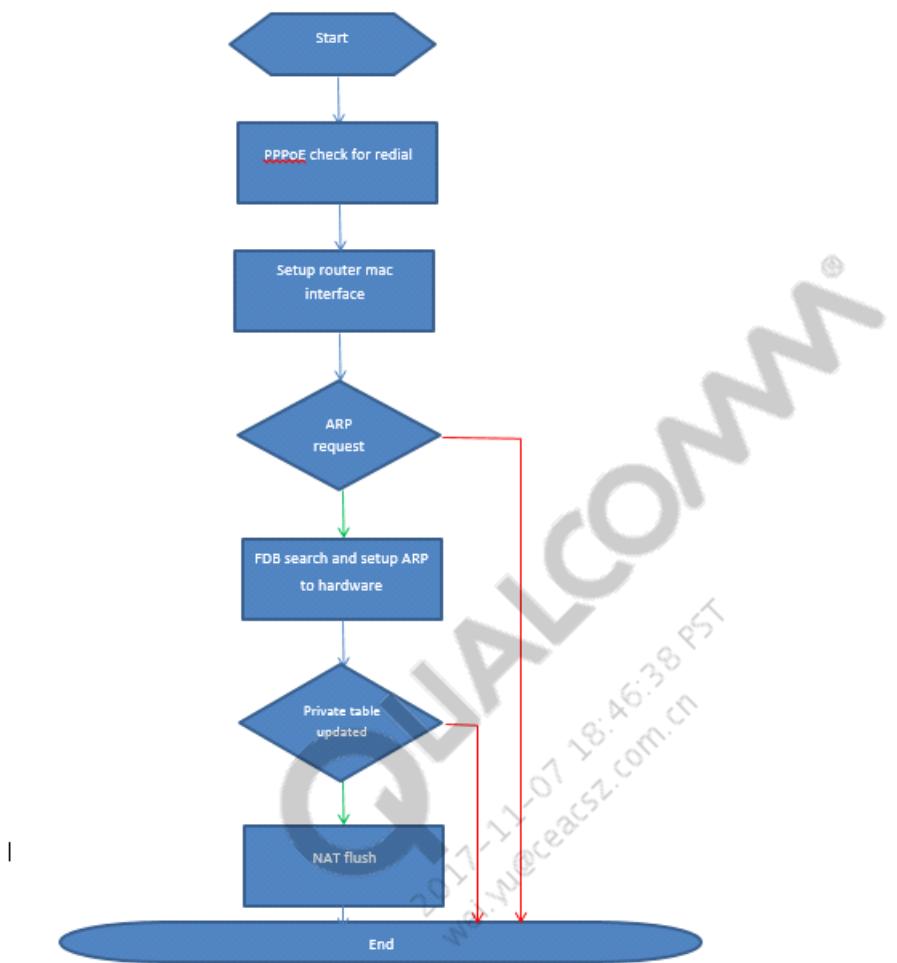


NatHelper is responsible for configuring the hardware NAT and NAPT entry for the UDP or TCP traffic offload. The following are the salient features of NatHelper:

- Configure the hardware arp table, router mac table and public table.
- Configure ACL rule to support PPPoe session acceleration.
- Support Proc file system for parameters read and write.
- Configure the hardware NAPT entry for specific five-tuple traffic flow for offload.
- Configure the hardware NAT entry by usage of Iptable command for offload.
- Connection management.
- Hardware table entry aged handle.

### 12.5.1 ARP packet process

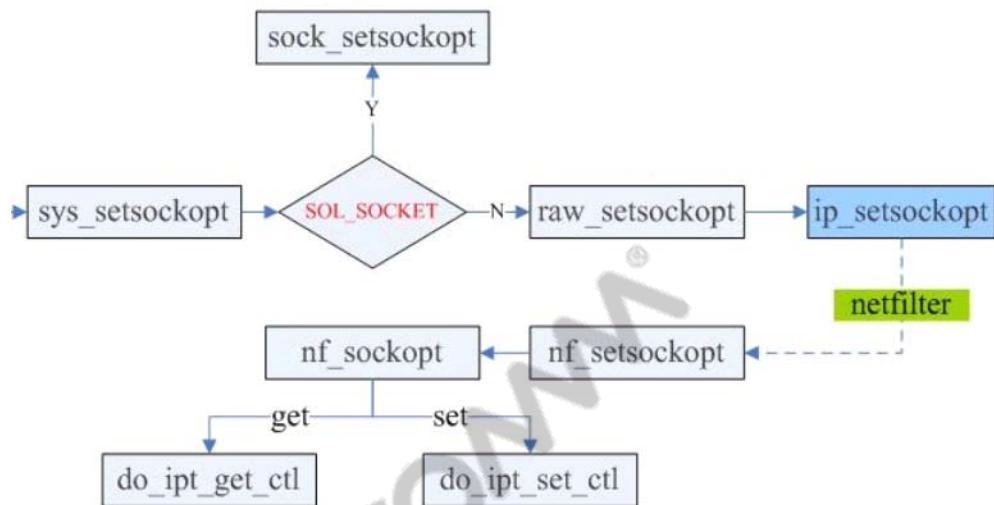
HostHelper registers the ARP hook in ARP\_IN with netfilter. It checks and handles the ARP packet.



WAN and LAN port information are obtained from DTS, and the VID configuration can be checked from forwarding database (FDB) entry table dynamically.

### 12.5.2 Static NAT implementation

NatHelper uses the iptable command to configure the hardware NAT function. The iptable command works as follows:



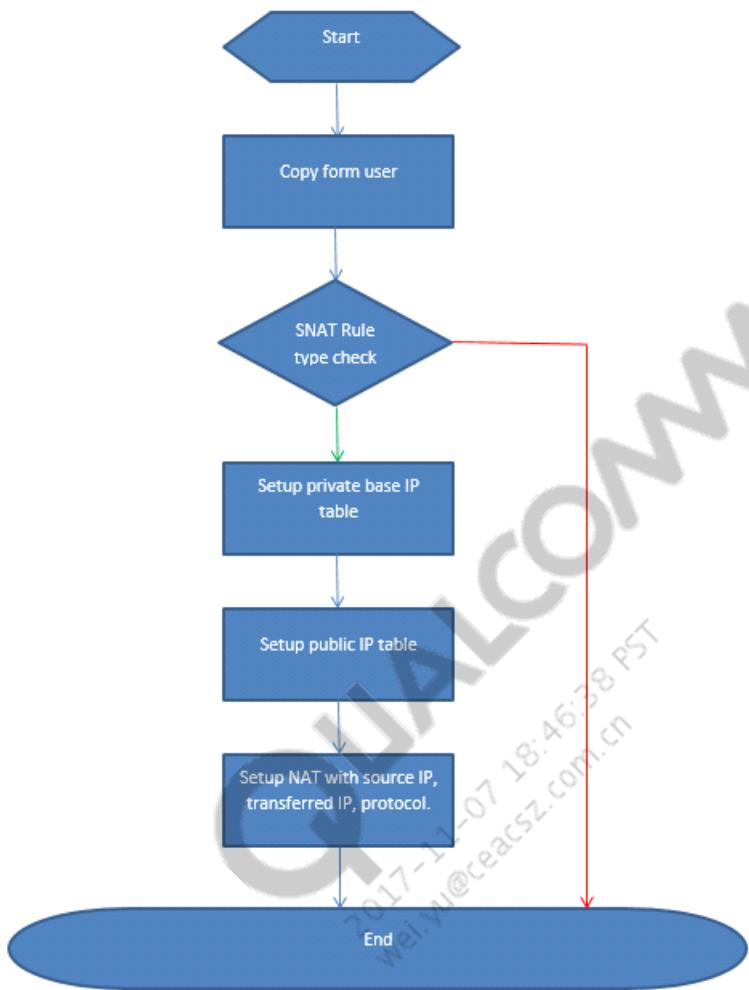
During initialization, Nathelper searches the netfilter sockopt, then replaces the default set/get handler, and the default sockopt set/get handler is saved.

If a user enters the following command, the new handler is called in this case. It handles SNAT type rule only; otherwise, it bypasses to the original saved opts handler.

```
iptables -t nat -A POSTROUTING -p tcp -s 192.168.11.10 -j SNAT --to 192.168.10.1
```

If this is the expected configuration, then this rule is configured to hardware with the source IP address, transferred IP address, and the protocol type UDP or TCP. This rule is effective for both SNAT and DNAT operations. Do not check the interface information in the iptables command.

The following flowchart illustrates the scenario for the hardware NAT entry configuration process:



### 12.5.3 NAPT implementation

NatHelper supports the following two modes:

Mode0: Limits to a maximum of up to 8 flow entries that can be offloaded by hardware. The related counters can be synchronized to the kernel stack.

Mode1: Support 1024 flow entries and do not synchronize the counters. It's the default mode.

#### 12.5.3.1 Connection manager

For connection management, it allocates a connection hash table. The size of the hash table is 8 times the size of the hardware NAPT entry number. For every connection, it simply uses the netfilter connection tracking data structure.

During the initialization phase, it registers the connection tracking event to kernel. When a new record is added, the callback is executed. The expected connection address is added to the local hash table if it matches all of the following conditions:

- NAT type
- TCP and established status or UDP
- Matches the private IP base address.

Also, the packet statistics of the record are added to the local hash table when the new netfilter connection tracking address is added.

### 12.5.3.2 Acceleration strategy

The period task classifies the five-tuple flow that can be offloaded to the hardware, and the time at which the flow can be offloaded. Due to hardware resource limitations, every connection is not offloaded. Every 5 seconds, the task checks the old connection. If the total number of packets in both directions during this period reaches the threshold, then this flow is chosen to be offloaded. Otherwise, the flow is retained in the table for next periodic check and the packet counter is updated with the latest packet flow that comes from the netfilter connection tracking.

The threshold initial value uses the base threshold value of 50. The threshold value is changed dynamically using the following formula:

```
packets_bdir_thres_new = ct_avg_pkts * (1 + (ct_offload_cnt/hw_cap));
```

If the latest calculated threshold value is greater than 50, this value is used as the traffic offload judge threshold; otherwise, the value of 50 is used as the traffic offload judge threshold.

Hw\_cap is the total hardware offload number. In mode0, up to 8 counters are supported; a total of up to 8 flows can be added to NAPT table. In mode1, up to 1024 entries are supported in the NAPT table.

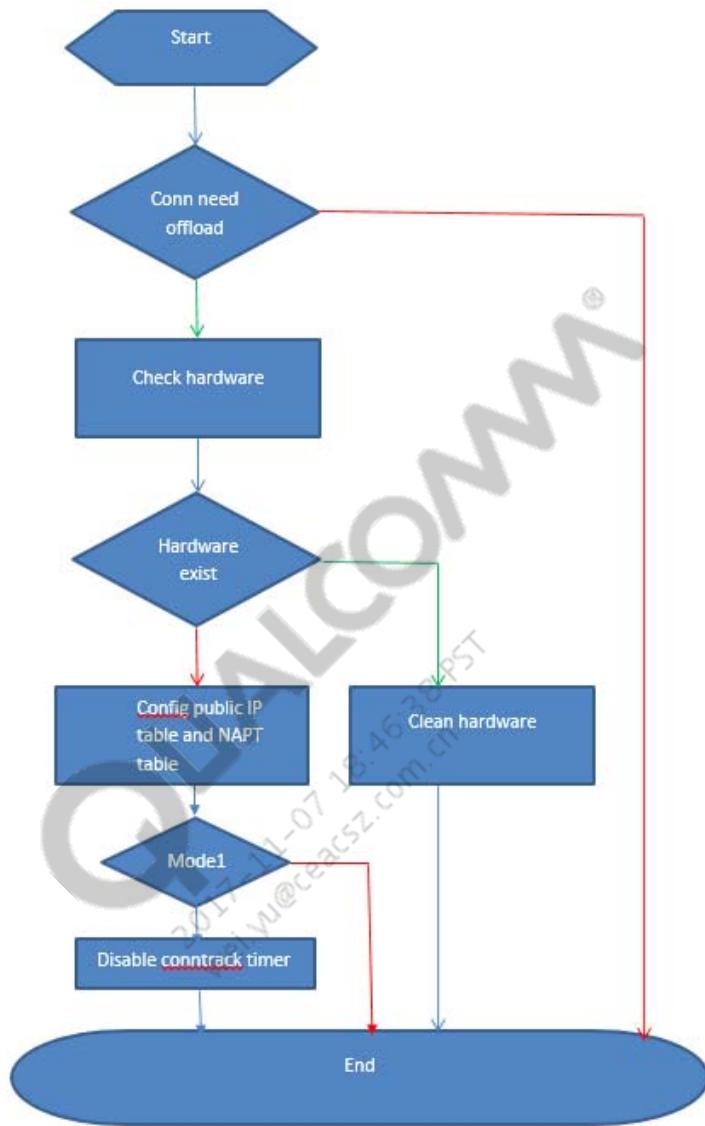
Two variable factors, namely, ct\_avg\_pkts and ct\_offload\_cnt, exist.

- ct\_avg\_pkts = all\_sessions\_packet\_count\_per\_5\_secs / session\_numbers.
- ct\_offload\_cnt is current NAPT entries in the hardware.

If traffic in a certain session is very low in an application, the base threshold can be adjusted to meet the requirement:

```
echo x > /proc/qca_switch/nf_athrs17_hnat_base_thresh
```

The following workflow illustrates the scenario to configure hardware while determining the connection must be offloaded:



### 12.5.3.3 Acceleration for PPPoE

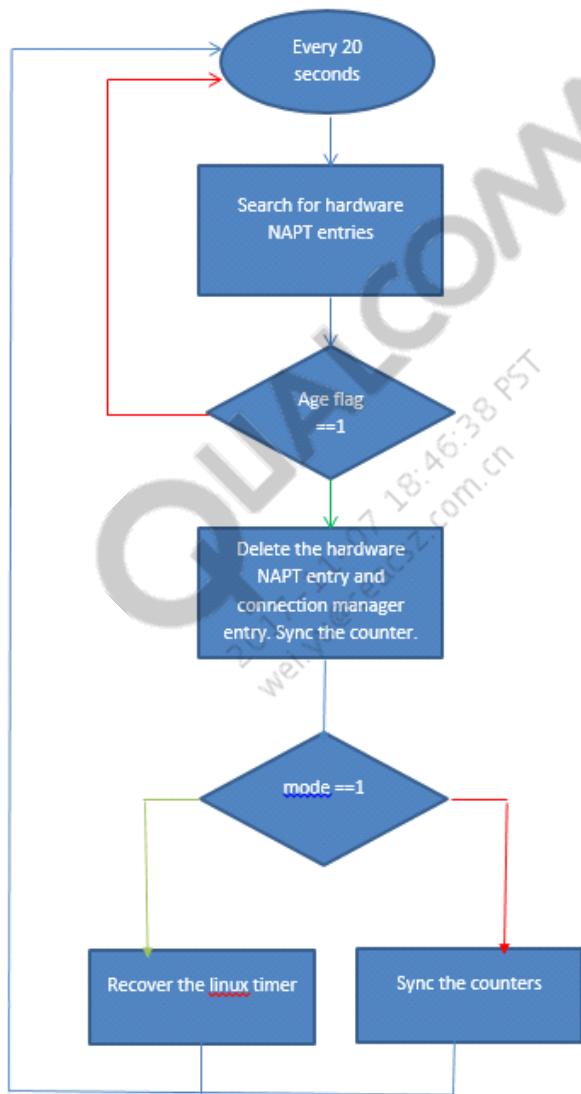
Two ACLs are applied for the PPPoE connection. The first ACL binds on the CPU port and forces the ARP\_INDEX\_OVER\_EN to the next hop (PPPoE peer/server). The second ACL binds on LAN ports, forcibly replaces the ARP\_INDEX to the next hop, and performs SNAT tasks if the destination IP address is not the LAN-side IP address.

1. Rule 1: For source IP addresses matched the WAN IP, the ACL forces their outgoing packets with the DMAC replaced by the next hop MAC (PPPoE peer MAC) and changes the CVID to the WAN port VID. This ACL applies on the CPU port for LAN to WAN packets.
2. Rule 2: For packets with destination IP outside of the LAN IP range, the ACL forces the outgoing packets with DMAC replaced by the next hop MAC (PPPoE peer MAC). SNAT

processing is also forced to be performed for those packets. This ACL is applied on LAN ports for packets from LAN to WAN. This ACL is also called the *default route ACL*.

#### 12.5.3.4 Hardware aging mechanism

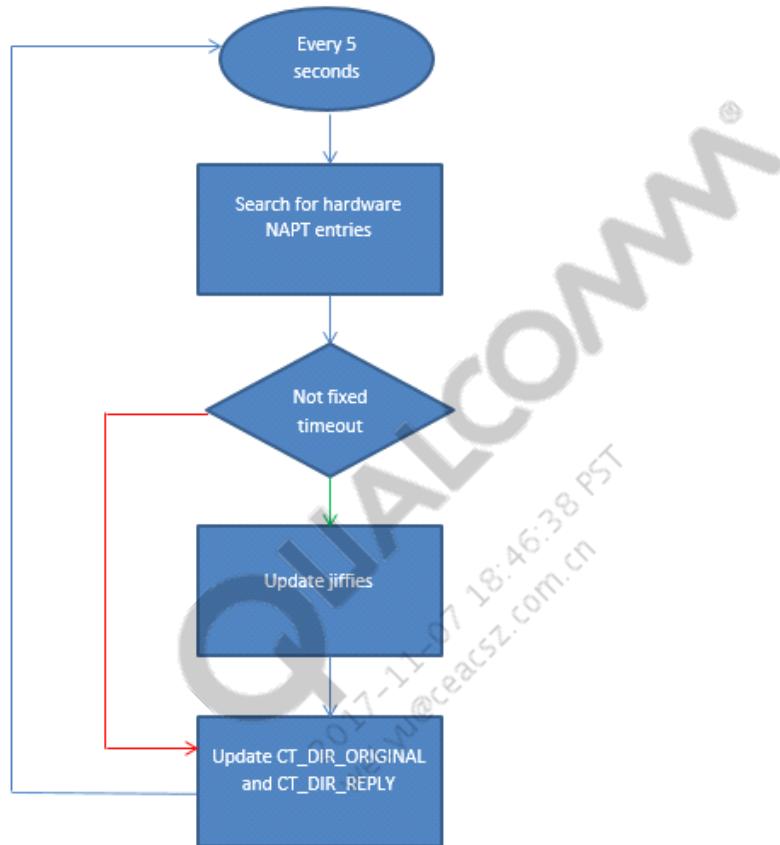
For the aged NAPT entries in hardware, NatHelper cleans them and also synchronizes the counter to Linux stack.



**NOTE** The hardware engine uses the age flag, which is a counter that decreases from the initial value of 6. Before the age expires, if the flow matches the entry again, the flag is restored to 6 by hardware engine.

### 12.5.3.5 Flow counter synchronization

For mode1, the period task searches the hardware NAPT entries and get the counters, then updates the jiffies if the connection is not a fixed timeout, and synchronizes the counters to Linux connection tracking.



### 12.5.3.6 SSDK shell and proc file debug

Perform the following steps for SSDK shell debug:

1. Check whether the NAT and NAPT are enabled.

```
ssdk_sh nat natstatus get
ssdk_sh nat naptstatus get
```

2. Print the router MAC address.

```
ssdk_sh ip intfentry show
```

3. Print the ARP entry.

```
ssdk_sh ip hostentry show
```

4. Print the static NAT entries.

```
ssdk_sh nat natentry show
```

5. Print the accelerated 5 tuple traffic.

```
ssdk_sh nat naptentry show
```

Perform the following steps for proc file system debug:

1. Enable or disable HNAT functions.

```
echo 1 > /proc/qca_switch/nf_athrs17_hnat
echo 0 > /proc/qca_switch/nf_athrs17_hnat
```

2. Set the scan timer of the task.

```
echo x > /proc/qca_switch/nf_athrs17_hnat_scan
```

3. Set the offload base threshold value.

```
echo x > /proc/qca_switch/nf_athrs17_hnat_base_thresh
```

### 12.5.3.7 Limitations

The following limitations exist with NatHelper:

- A total of 8 flow counters are supported at a point in time. The counter only includes ingress/egress bytes and packets for the flow. No TCP traffic window-related statistics are computed.
- Fragment packets are not offloaded.

### 12.5.4 Sample configuration scenario of static and dynamic NAT

Consider a sample configuration scenario for verifying the working of static NAT and dynamic NAPT. The following is the PC configuration:

- LAN PC: ipaddress 192.168.11.10, gateway 192.168.11.1
- WAN PC: ipaddress 192.168.10.10, gateway 192.168.10.1

1. Configure the interface as follows:

```
ifconfig eth0 192.168.10.1
ifconfig eth1 192.168.11.1
```

2. Enter the following command to enable the HNAT module:

```
ssdk_sh nat global set enable disable (it is recommended to use this mode)
```

This is the default mode. The following mode offloads only up to a maximum 8 flows:

```
ssdk_sh nat global set enable enable
```

If WAN port is not 5, for example if the WAN port is 1, enter the following command to change the WAN port bitmap:

```
echo 2 > /sys/ssdk_napt/wan_port
```

3. Perform a ping to the LAN PC:

```
Ping 192.168.11.10
```

4. Perform a ping to the WLAN PC:

```
Ping 192.168.10.10
```

5. Enter the following command to configure the NAT table:

```
iptables -t nat -A POSTROUTING -p udp -s 192.168.11.10 -j SNAT --to  
192.168.10.1
```

When UDP traffic is sent, it must be offloaded.

6. Enter the following command to track SNAT:

```
iptables -A POSTROUTING -t nat -o eth0 -j MASQUERADE
```

7. Enter the following command to track DNAT:

```
iptables -I PREROUTING -t nat -i eth0 -p udp --dport 2000:3000 -j DNAT  
--to 192.168.11.10 -m state --state NEW
```

When UDP traffic is sent, it must be offloaded 5~10 seconds later.

Do not configure NAT while verifying the working of NAPT.

# 13 Beacons and Frames Transmission

---

This chapter describes the functionalities supported in the transmission of beacons, probe request frames, association frames, and probe response frames. Mechanisms supported for the transmission of beacons and frames, such as, Quick STA Kickout (QSK), Software Retry (SWR), bursted beaconing, Dynamic Transmit Chainmask Selection (DTCS), HotSpot 2.0, TX99 Mode, Raw Mode Tx/Rx, Wake on Wireless AP Assist, Host Tx Flow Control, Descriptor configuration, dynamic beacon, are discussed in this chapter.

## 13.1 Software Retry (SWR)

Software Retry (SWR) is a mechanism to allow software to re-transmit one frame, in addition to hardware transmissions.

In the Qualcomm Technologies legacy Fusion driver, a frame (either non-AMPDU or AMPDU) is put into the hardware queue for transmission, and might be transmitted multiple times until a success occurs with different rates (that is, hardware retry) depending on the TX descriptor set by the Rate Adaptation algorithm. After the hardware finishes transmitting a frame (either success or failure), it will raise an interrupt to notify the software, which then performs any post-processing of this frame:

- For a successful frame, software completes the frame and releases the buffer.
- For a failed non-AMPDU frame: software completes the frame and releases buffer.
- For a failed AMPDU frame: In case of aggregation, the failed frame in the aggregate will be put back into the TIC queue for further transmissions, which is handled by the Block-ACK mechanism.

SWR retries the failed non-AMPDU frame (also called SWR eligible frame) at the software level; that is, putting it back in the TID queue again.

SWR is especially useful in the following scenarios:

- When the STA transitions from active mode to PS mode, there might be frames already in the hardware queue that will be lost if STA goes to sleep suddenly. In such a case, these lost frames should be retried later (placing them in the hardware queue again) instead of completing them and releasing the buffer.
- When a channel is bad or there is sudden interference, SWR can give a frame more chances to go through, especially when the application is sensitive to frame loss but can tolerate with frame delays.

### 13.1.1 Theory of Operation

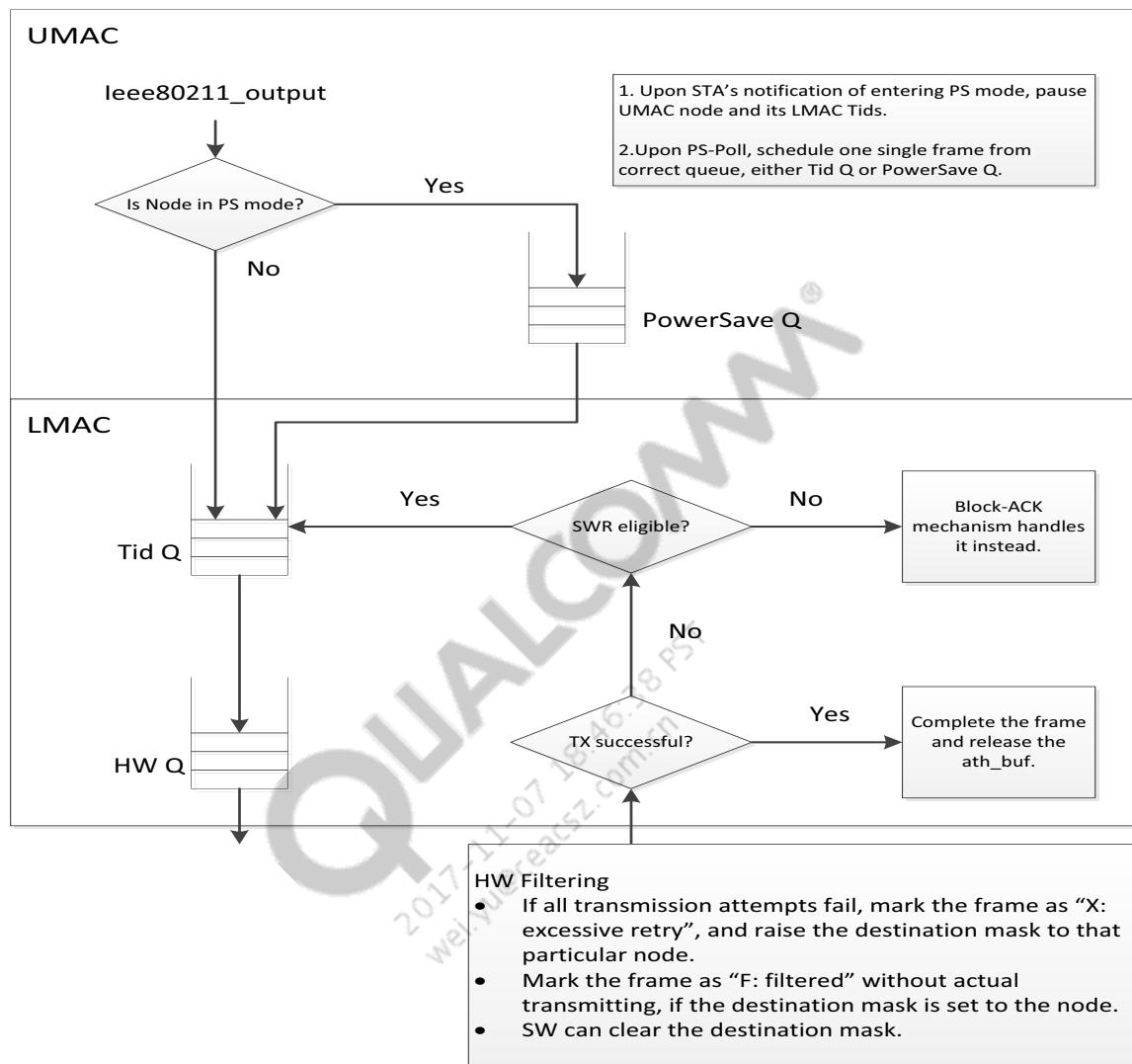
The operation of a system with SWR capability behaves in the following manner (refer to Overview of Software Retry).

There are five SWR function units in the system:

- Hardware filtering
- Requeue of non-AMPDU frame
- Pause the TID in PS mode
- Response to PS-Poll
- TIM and MORE bit setting

#### 13.1.1.1 Hardware Filtering

By default, hardware has the capability of filtering a certain frame destined to a particular destination node. If a frame (to node A for example) fails due to excessive retry, hardware will set the destination mask of destination A, and any frame to A will be filtered out by hardware without even transmitting them. It is totally dependent on the software to determine when to clear the destination mask (each frame carries a flag). In the Qualcomm Technologies legacy driver, hardware filtering is enabled but does not actually take effect, since all frames enable that special flag and always clear the destination mask.

**Figure 13-1 Overview of Software Retry**

With SWR enabled, a mechanism of setting and clearing the destination mask is provided. Consider the following example where Tid 1 is for node A and Tid 2 is for node B:

**Figure 13-2 Example of Hardware Filtering**

With SWR enabled, if frame #1 fails due to excessive retry (marked as X), frames #2 and #3 will be simply filtered out (marked as F). To avoid an out of sequence error, there is a wait until all three frames have been requeued to Tid Q 1, and then the software starts scheduling a frame from

Tid Q 1 to hardware Q with the clear destination mask flag ON. Note that Tid Q 2 is not affected since it is for a different destination.

### 13.1.1.2 Requeue of non-AMPDU frames

Instead of completing the frame immediately after hardware finishes the transmissions, SWR requeues it to the corresponding Tid Q for future scheduling. The frame is inserted into the Tid Q by sequence number.

### 13.1.1.3 Pause the Tid in PS mode

Upon a STA notification of entering PS mode, the AP pauses the UMAC node (if UMAC node is paused, any incoming frame to this node will be buffered in the PowerSave Q of this node). Meanwhile, it also pauses all Tid Qs in the LMAC of this node. When the STA exits the PS mode, the AP should unpause both LMAC Tid Qs as well as the UMAC node.

### 13.1.1.4 Response to PS-Poll

Upon a PS-Poll from a PS STA, the AP schedules only one frame and sends it to the STA. When software retry is enabled, there might be frames buffered in UMAC PowerSave Q as well as LMAC Tid Q. Hence caution should be used when sending a single frame in this case.

### 13.1.1.5 Setting of TIM and MORE bit

When there are chances of coexistence of frames in PowerSave Q and Tid Q during PS mode, the AP must properly set the TIM in the beacon and MORE bit in each frame for transmission. For the TIM, AP clears the TIM bit when both queues are empty. For the MORE bit, the AP checks whether PowerSave Q is empty when transmitting frame from the Tid Q.

## 13.1.2 Implementation

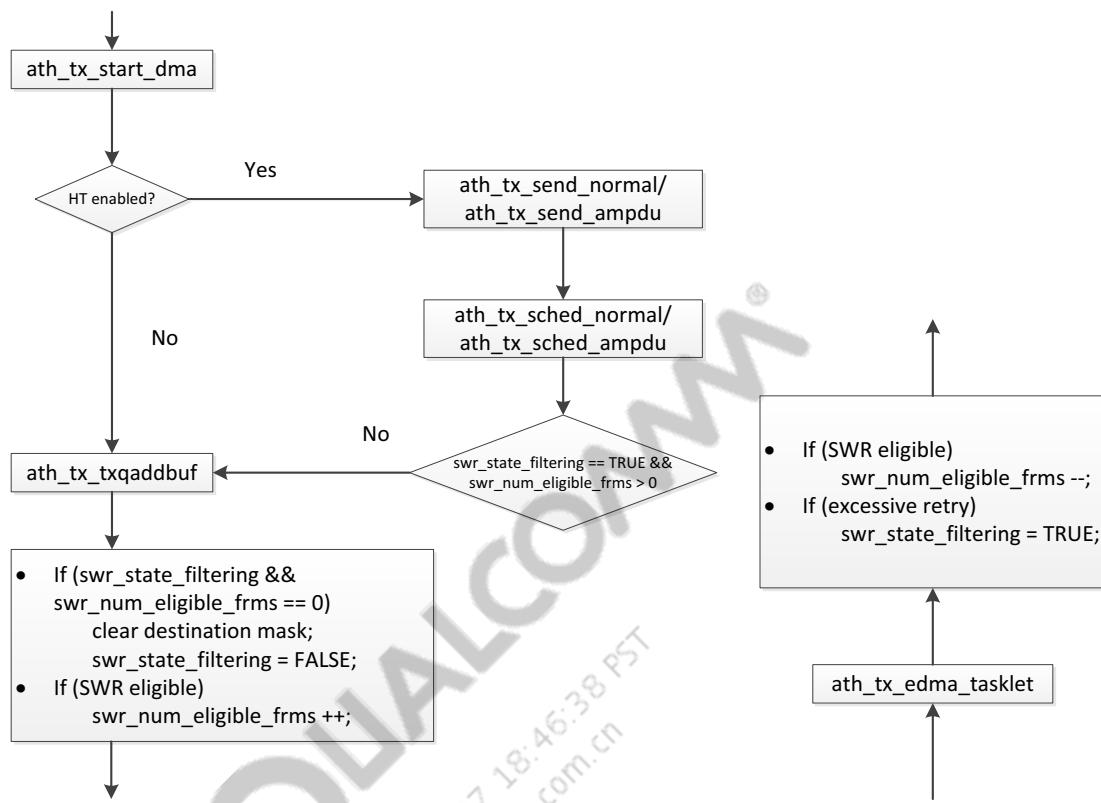
Implementation of SWR functions are described in the following subsections.

### 13.1.2.1 Hardware Filtering

Two variables (for each `ath_node`) are maintained for this purpose:

- `swr_state_filtering` is a Boolean value indicating whether the hardware Q is in filtering status or not. It is set to TRUE when a frame fails due to an excessive retry. It is cleared when there is no SWR eligible frame left in the hardware Q.
- `swr_num_eligible_frms` indicates the number of SWR eligible frames in hardware Q.

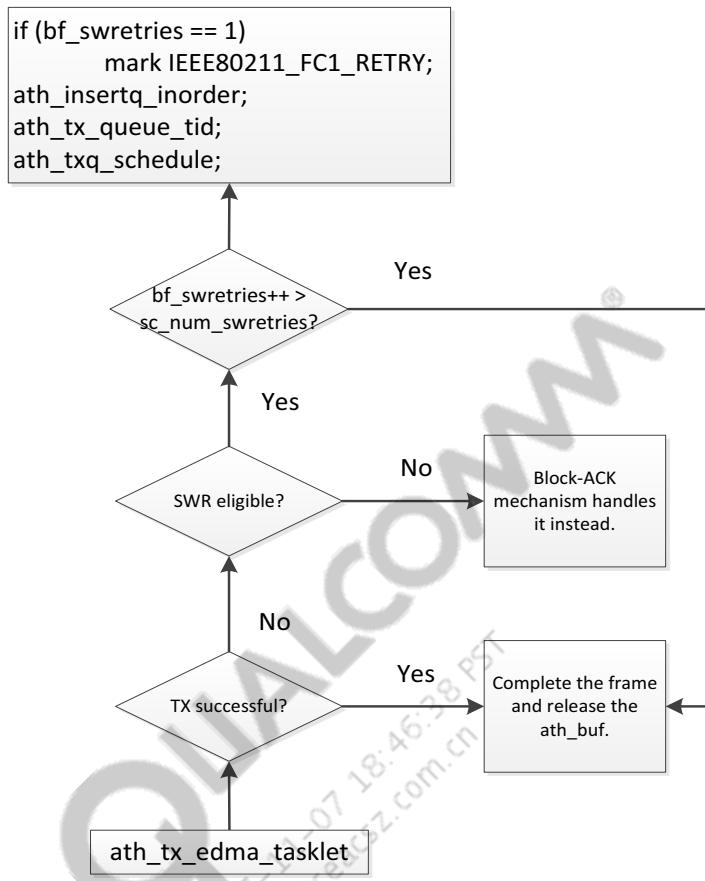
The Hardware Filtering flow is shown in [Figure 13-3](#).



**Figure 13-3 Implementation of Hardware Filtering Logic**

### 13.1.2.2 Requeue of non-AMPDU frame

- Requeuing of non-AMPDU frame is done in `ath_tx_mpdu_requeue`, which is shown in [Figure 13-4](#). There is a tunable parameter `sc_num_swretries`, which is the maximum SWR count allowed. The default value of `sc_num_swretries` is set to 2.

Figure 13-4 Flow of `ath_tx_mpdu_requeue`

### 13.1.2.3 Pause the Tid for PS Mode STA

When the AP receives notification from the STA that it is going into PS mode, the AP pauses the TID related to that node, in addition to pausing the UMAC node. It is achieved by calling the `ic_node_psupdate` API provided by LMAC. The implementation is in `ieee80211_mlme_ap.c`.

### 13.1.2.4 Response to PS-Poll

When the node is in PS mode, its TIDs are already paused. Hence, the AP does not send a DATA frame out upon PS-Poll reception. In such a case, a variable `an_pspoll_pending` is maintained for each `ath_node`, and an API `ic_handle_pspoll` to UMAC is provided:

- `an_pspoll_pending`. It is TRUE if there is a PS-Poll received from the PS STA and a reply is needed.
- `ic_handle_pspoll`. UMAC calls this API to try to schedule a frame from the Tid Q. It returns a non-zero value if no frame is transmitted by LMAC.

The flow of handling the PS-Poll is shown in [Figure 13-5](#) and [Figure 13-6](#).

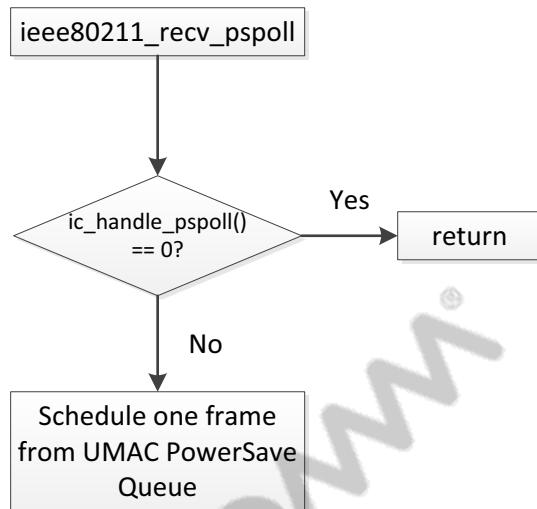


Figure 13-5 Handling PS-Poll (UMAC Part)

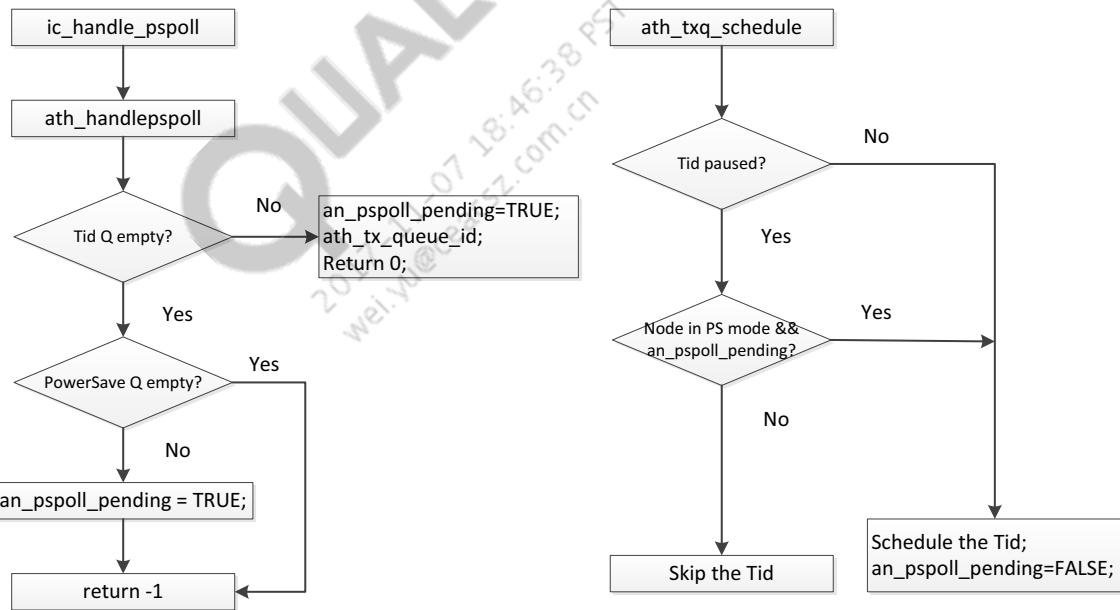


Figure 13-6 PS-Poll Handling (LMAC Part)

### 13.1.2.5 Setting TIM and MORE bit

The UMAC and LMAC exchange PowerSave Q and Tid Q length information, in order to set TIM and the MORE bit correctly. The following APIs are provided:

- `sc_ieee_ops->get_pwrsaveq_len`. Returns the length of the PowerSave Q.
- `ic->ic_exist_pendingfrm _tidq`. Returns whether there is any frame in Tid Q.

## 13.1.3 Configuration

### 13.1.3.1 Enable SWR

To enable SWR, the following flags need to be set to 1:

- ATH\_SWRETRY in wlan/os/linux/BuildCaps.inc
- ATH\_SWRETRY in wlan/hal/linux/Makefile.inc

Also in ath\_dev/Kbuild, ath\_swretry.o should be included into the target.

### 13.1.3.2 Debug SWR

To debug SWR, there is a debug level for SWR: ATH\_DEBUG\_SWR. To enable SWR related debug prints, use the following command:

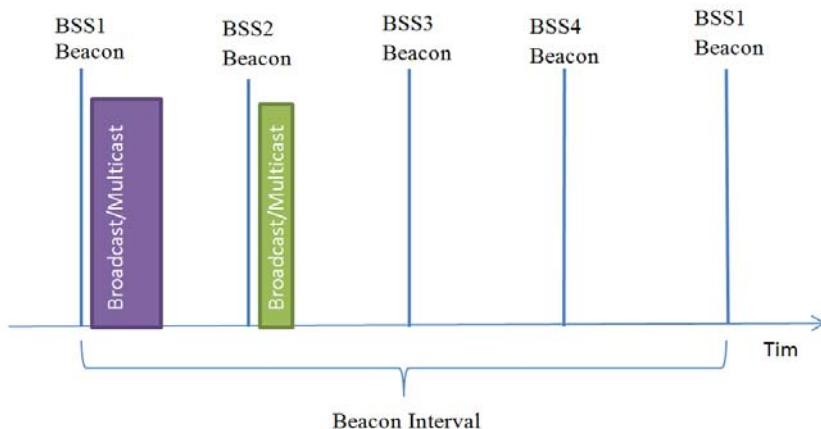
```
iwpriv wifi0 ATHDebug ATH_DEBUG_SWR (for example, 0x10000000)
```

To reset the debug level, use:

```
iwpriv wifi0 ATHDebug 0
```

## 13.2 Bursted Beaconing

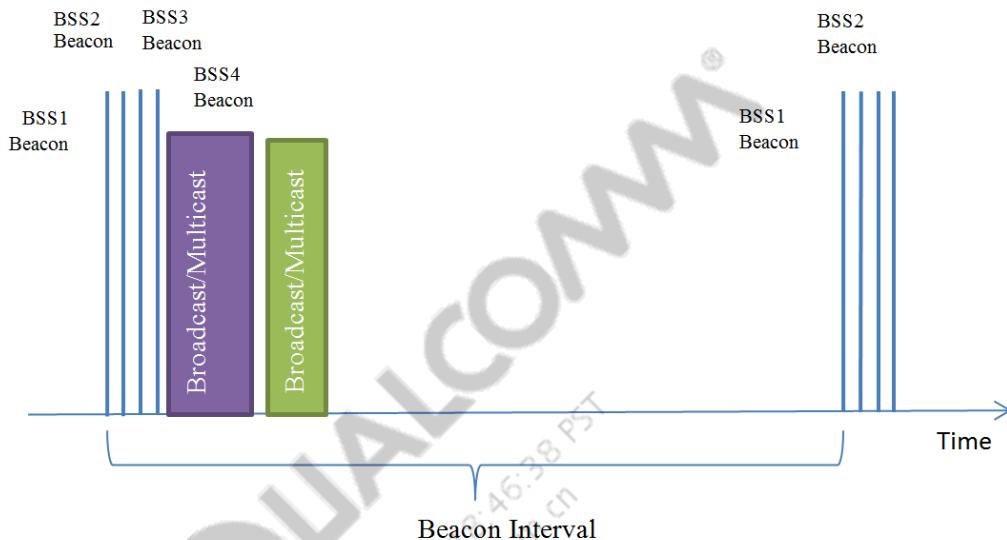
The default beaconing scheme when multiple BSSes are active on the same radio is *staggered beaconing* (see [Figure 13-7](#)). With 4 active BSSes, beacon interval of 100 TUs and DTIM of 1, beacons for the BSSes are staggered in time and are 25 TUs (= 100 TUs / 4) apart. A staggered beaconing scheme requires TSF Offset capability in the hardware, and all but some of the first generation MAC chips have this capability.



**Figure 13-7 Staggered Beaconing Scheme**

An alternate beaconing scheme is *bursted beaconing* (see [Figure 13-6](#)). In this scheme, beacons for all BSSes are transmitted together one after another. As multicast and broadcast data frames follow DTIM beacons in the order of their respective BSSes, STAs associated to higher BSSes are

progressively worse off in terms of power consumption as they stay awake longer after the beacon. A STA associated to BSS2 must stay awake during the time multicast and broadcast data for BSS1 is transmitted. On the other hand, a STA associated to BSS3 or BSS4 can immediately go back to sleep as BSS3 and BSS4 beacons indicate in their TIM that there is no multicast data to be transmitted for these BSSes.



**Figure 13-8 Bursted Beaconing Scheme**

The beaconing scheme is stored in `sc_stagbeacons` and is set to 1 and 0 respectively for staggered and bursted beaconing. This parameter is set prior to activating the BSSes. Beacon interval for all BSSes are assumed to be identical. It is feasible to modify this behavior to support arbitrary beacon intervals for BSSes with additional functionality.

Software Beacon Alert (SWBA) interrupts kickstart beacon processing. These interrupts occur a few milliseconds prior to Target Beacon Transmit Time (TBTT). SWBA interrupts are handled in an interrupt context instead of in a tasklet to prevent latencies in beacon processing. Staggered beaconing requires more frequent SWBA interrupts and hence have a higher computational overhead over bursted beaconing. On an SWBA interrupt, the contents of the beacon frame for the BSS are updated and the beacon frame is queued to the hardware beacon queue. If it is a DTIM beacon and there are multicast/broadcast frames for the BSS waiting to be transmitted, they are queued to a second hardware queue called the CAB (content after beacon) queue. The CAB queue is configured to be gated by beacon queue so that frames on the CAB queue are suspended for transmission until the beacon queue is empty. The beacon queue itself is gated by a time value, TBTT, so that beacon frames are sent out only at TBTT. In the case of bursted beaconing, multicast/broadcast frames from all BSSes are concatenated and queued on the CAB queue on a DTIM SWBA.

## 13.3 Wake on Wireless AP Assist

This feature is used to send wake on wireless magic packet to any associated client to the access point. Whenever STA receives a magic packet, it wakes up the host.

### Magic packet

The magic packet is a broadcast frame containing anywhere within its payload 6 bytes of all 255 (FF FF FF FF FF FF in hexadecimal), followed by sixteen repetitions of the target computer's 48-bit MAC address, for a total of 102 bytes. Because the magic packet is only scanned for the string above, and not actually parsed by a full protocol stack, it may be sent as any network and transport-layer protocol, although it is typically sent as an UDP datagram to port 7 or 9, or directly over Ethernet as EtherType 0x0842.

Whenever the user /application wants to send magic packet to a particular connected node, an iwpriv/ioctl is issued to the access point. The Wi-Fi driver prepares magic packet for that node and sends that packet as QOS data frame.

### 13.3.1 High Level Design

The magic packet is prepared in the UMAC layer with a proper Ethernet header and with the magic packet payload. The magic packet is transmitted as QOS data packet with WME\_AC\_BK.

#### Driver Code:

```

/* Initialize */
wbuf_append(mywbuf, msg_len);
memset((u_int8_t *)wbuf_raw_data(mywbuf), 0, msg_len);
/* Prepare Payload */
payload = (u_int8_t *)wbuf_raw_data(mywbuf);
/* Copy sync */
for (index = 0; index < 6 ; index++) {
    payload[index] = 0xff;
}
for ( i=0 ; i < 16; i++) {
    IEEE80211_ADDR_COPY(&payload[index], &macaddr);
    index +=6;
}
/* Prepare ethernet packet */

eh = (struct ether_header *) wbuf_push(mywbuf, sizeof(struct ether_header));
IEEE80211_ADDR_COPY(&eh->ether_shost[0], src_mac_address);
IEEE80211_ADDR_COPY(&eh->ether_dhost[0], macaddr);
eh->ether_type = htons(0x0842); /* WOL type */

/* prepare wbuf for sending */
mywbuf->dev = netd;
wbuf_set_priority(mywbuf,WME_AC_BK);
wbuf_set_tid(mywbuf, WME_AC_TO_TID(WME_AC_BK));
skb->dev = dev;
dev_queue_xmit(skb);

```

### 13.3.2 WoW packet format on Wire Shark

Figure 13-9 shows the WoW packet format on wire shark.

```
No.      Time           Source          Destination        Protocol Length Info
8421 18.579323 00:34:56:78:e5:e5 IntelCor_5e:0c:64      WOL      280  MagicP
      acket for IntelCor_5e:0c:64 (00:24:d6:5e:0c:64)

Frame 8421: 280 bytes on wire (2240 bits), 280 bytes captured (2240 bits)
Prism capture header
IEEE 802.11 QoS Data, Flags: .....F.
Type/Subtype: QoS Data (0x28)
Frame Control: 0x0288 (Normal)
Duration: 44
Destination address: IntelCor_5e:0c:64 (00:24:d6:5e:0c:64)
BSS Id: 00:34:56:78:e5:e5 (00:34:56:78:e5:e5)
Source address: 00:34:56:78:e5:e5 (00:34:56:78:e5:e5)
Fragment number: 0
Sequence number: 11
QoS Control
Logical-Link Control
DSAP: SNAP (0xaa)
IG Bit: Individual
SSAP: SNAP (0xaa)
CR Bit: Command
Control field: U, func=UI (0x03)
Organization Code: Encapsulated Ethernet (0x000000)
Type: Wake on LAN (0x0842)
Wake On LAN, MAC: Intelcor_5e:0c:64 (00:24:d6:5e:0c:64)
Sync stream: ffffffffffffff
MAC: IntelCor_5e:0c:64 (00:24:d6:5e:0c:64)
```

Figure 13-9 WoW Packet Format

## 13.4 Dynamic Transmit Chainmask Selection (DTCS)

Dynamic Transmit Chainmask Selection is a feature that allows the transmit chainmask to be changed dynamically at frame transmission time. One of the applications of Dynamic Transmit Chainmask Selection is to help avoid the degradation of EVM that happens because of frequency notching at high modulation rates when single stream frames are transmitted on multichain devices.

### 13.4.1 Theory of Operation

When a single stream frame is transmitted, each antenna in a multichain configuration transmits the same signal. The transmissions on each antenna are delayed from each other. The reason for this is to prevent unintentional beamforming. However this causes frequency notching, which increases the transmit EVM. The degradation is most prominent for higher modulation rates (64 QAM) which have the highest EVMs. To combat this effect, single stream transmissions at high modulation rates are transmitted on only one chain. This mechanism is called dynamic transmit chainmask selection. Transmitting on a single chain eliminates the frequency notching.

This feature is not used if the WLAN hardware using transmit beamforming to send the frame to another STA.

## 13.4.2 Implementation

The feature is implemented partly in the LMAC and partly in the rate control module. The LMAC module queries the rate control module to determine the rate to use for transmitting a frame. It again queries the rate control module to find the number of chains to be used for transmission. The rate control table has a column that specifies the chainmask for each rate. The LMAC applies the returned chainmask to the descriptor of the frame to be sent.

The number of transmit chains will be written to the descriptor fields and passed by the MAC to the baseband, where it is used to select the number of chains to use for transmission.

### 13.4.2.1 Rate Control Interface

`ath_rate_max_tx_chains()`:

- Input: Rate, Rate flags
- Output: Number of chains to be used for transmission

## 13.4.3 Configuration

This feature is turned off at build time for most platforms. For select platforms, it is turned on at build, but is turned off at runtime. At runtime the feature is turned on using the following command:

```
iwrpriv wifi0 dyntxchain 1
```

At runtime, the status (on/off) of the feature can be checked using:

```
iwpriv wifi0 get_dyntxchain
```

## 13.5 Quick STA Kickout (QSK)

Quick STA Kickout is a feature that allows the Access Point to disassociate an STA if the AP believes that it has lost connectivity to the STA. The method by which it detects loss of connectivity is through excessive retries of frames transmitted to that STA.

### 13.5.1 Theory of Operation

Loss of connectivity between an AP and a STA can impact other STAs in the vicinity, under certain conditions. For example, if a UDP video stream from a media server is flowing between an AP and a STA, and the STA loses connectivity (for example, the wireless card is removed, or the system crashes), the video stream will continue to be transmitted for a significant period of time by the media server. Because the STA has lost connectivity, each frame will be retransmitted several times by the AP. Because UDP does not have a feedback mechanism (unlike TCP), it may be some time before the media server realizes that the end point is no longer responding and stops streaming. The streaming frames and their retransmissions consume a significant amount of airtime, impacting other STAs in the vicinity. Quick STA Kickout attempts to minimize the impact of this condition by disassociating a STA if the AP thinks it has lost connectivity to the STA. Once

the STA is disassociated, incoming UDP frames from the media server are dropped at the AP, instead of being sent on the air.

### 13.5.2 Implementation

The feature is implemented in the UMAC. On completion of a transmission, the software checks to see if the frame has been excessively retried and has failed transmission. If the transmission fails, software increments a counter. The counter is reset when a frame is successfully transmitted. If the counter exceeds a threshold (default is 50), the STA is disassociated.

The UMAC function `ieee80211_complete_wbuf()` calls `ieee80211_kick_node()` if the counter `ni_consecutive_retries` exceeds a threshold.

### 13.5.3 Configuration

This feature is turned on at build time for most platforms. For select platforms, it is turned off at build time. It cannot be turned off at runtime. At runtime the threshold can be configured using the following command:

```
iwrpriv ath0 sko <threshold>
```

The value of the threshold can be read using

```
iwpriv ath0 get_sko
```

## 13.6 Support for radiotap header

Currently, radiotap is considered as an industry standard. Linux kernel and other chip manufacturers migrated from PRISM to radiotap many years back. In the future, only radiotap is planned to be supported. However, because of the present infrastructure requiring support for PRISM, the transformation to radiotap is not entirely completed. During this period of migration from PRISM to radiotap, both PRISM and radiotap are continued to be supported with an early stage of radiotap format.

The radiotap.org website specifies radiotap as an industry defacto standard for 802.11 frame injection and reception. The radiotap header format provides more flexibility than the Prism or AVS header formats and allows the driver developer to specify an arbitrary number of fields based on a bitmask presence field in the radiotap header.

The radiotap capture format starts with a radiotap header:

```
struct ieee80211_radiotap_header {
    u_int8_t      it_version;      /* set to 0 */
    u_int8_t      it_pad;
    u_int16_t     it_len;          /* entire length */
    u_int32_t     it_present;      /* bitmask of the radiotap data
fields that follows the radiotap header. */
} __attribute__((__packed__));
```

The following are the standard radiotap fields in one `u_int32_t` mask:

| bit number | field                            |
|------------|----------------------------------|
| 0          | /TSFT                            |
| 1          | /Flags                           |
| 2          | /Rate                            |
| 3          | /Channel                         |
| 4          | /FHSS                            |
| 5          | /Antenna signal                  |
| 6          | /Antenna noise                   |
| 7          | /Lock quality                    |
| 8          | /TX attenuation                  |
| 9          | /dB TX attenuation               |
| 10         | /dBm TX power                    |
| 11         | /Antenna                         |
| 12         | /dB antenna signal               |
| 13         | /dB antenna noise                |
| 14         | /RX flags                        |
| 19         | /MCS                             |
| 20         | /A-MPDU status                   |
| 21         | /VHT                             |
| 22-28      | reserved                         |
| 29         | /Radiotap Namespace              |
| 29 + 32*n  | /Radiotap Namespace              |
| 30         | /Vendor Namespace                |
| 30 + 32*n  | /Vendor Namespace                |
| 31         | reserved: another bitmap follows |
| 31 + 32*n  | reserved: another bitmap follows |

### Linux kernel radiotap implementation

In general, WLAN driver provides all necessary information through struct ieee80211\_hw, then Linux kernel will use ieee80211\_rx\_monitor()&ieee80211\_add\_rx\_radiotap\_header() to add radiotap header for all packets from monitor VAP. Related structures are struct ieee80211\_hw, struct ieee80211\_conf, struct wiphy.

The following is the call stack in case of ath10k:

```

ath10k_htt_t2h_msg_handler=>
ath10k_htt_rx_handler=>
ath10k_htt_rx_msdu=>
ath10k_process_rx=>
ieee80211_rx(ar->hw, skb);
==> skb = ieee80211_rx_monitor(local, skb, rate);

```

```
==> ieee80211_add_rx_radiotap_header()
```

A new iwpriv command is added so that user can choose the header format to use, prism or radiotap. Radiotap is planned to be the primary supported header in the future. The prism header will be deprecated gradually.

Use the following iwpriv commands added to switch between radiotap (default) and prism header for monitor VAP:

```
iwpriv ath01 mon_decoder 1 to use prism  
iwpriv ath01 mon_decoder 0 to use radiotap
```

### Data structure changes and additional information in WLAN driver

Because all the necessary structures, such as struct ieee80211\_hw, struct ieee80211\_conf, struct wiphy, are GPL licensed, they are not directly used in QCA proprietary driver and are rewritten.

Most of the following radiotap defined fields will be populated in driver.

- defined-fields/A-MPDU status
- defined-fields/Antenna
- defined-fields/Antenna noise
- defined-fields/Antenna signal
- defined-fields/Channel
- defined-fields/FHSS
- defined-fields/Flags
- defined-fields/Lock quality
- defined-fields/MCS
- defined-fields/RX flags
- defined-fields/Radiotap Namespace
- defined-fields/Rate
- defined-fields/TSFT
- defined-fields/TX attenuation
- defined-fields/VHT
- defined-fields/Vendor Namespace
- defined-fields/dB TX attenuation
- defined-fields/dB antenna noise
- defined-fields/dB antenna signal
- defined-fields/dBm TX power

Some of the following radiotap suggested fields are populated in driver:

- suggested-fields/RSSI
- suggested-fields/RTS retries

- suggested-fields/TX flags
- suggested-fields/XChannel
- suggested-fields/data retries
- suggested-fields/extended flags
- suggested-fields/hardware queue
- suggested-fields/timestamp

Although FCS is a rejected-field in radiotap official document, this field is added in radiotap Vendor Namespace.

Most of the fields are supported by open source wireshark. However, certain new or private fields, such as the fields related to MU-MIMO, are added in wireshark.

A new file os/linux/src/osif\_mon.c added for the monitor header formats, namely, radiotap and prism. Original prism codes are moved to the file and optimized. The umac/include/ieee80211\_radiotap.h for most of radiotap macros and definitions are reused.

New macros are defined for new standard 802.11AC. Unified and cleaned up the current PRISM implementation, API optimized in monitor Rx function osif\_receive\_monitor\_80211\_base(). A new function, osif\_mon\_add\_radiotap\_header(), is implemented for radiotap support in the new file osif\_mon.c

The following fields might not be accurate:

- MAC timestamp (MAC timestamp radiotap header field, "radiotap.mactime", indicates the arrival time of the last bit of the AMPDU per radiotap header definition. In the AMPDU case, all subframes except the last subframe had their radiotap.mactime set to 0xFF. Similarly, all PPDU frames have the same radiotap.mactime value, which is the arrival time of the last bit of the PPDU.)
- SSI Signal (it is in RSSI value instead of in dbm)
- VHT 80+80 parsing

The following fields are not present:

- Antennas
- AMPDU details
- FCS

### 13.6.1 Radiotap changes for PCI device ID and antenna DB in sniffer feedback

Starting with QCA\_Networking\_2017.SPF.5.0, an enhancement is implemented to enable users ascertain the radio from which the capture is received to make certain calculations and analysis in post-processing. The PCI device ID is to be added to Radiotap vendor namespace. Also a separate field, Antenna DB was added to the Radiotap header. No change is implemented in the way mactime is recorded in the Radiotap header. Customers can identify this value in their post-processing scripts.

The following is a sample of the sniffer feedback recoded after capturing frames in monitor mode and transferring to a PC running wireshark

```

Radiotap Header v0, Length 44
  Header revision: 0
  Header pad: 0
  Header length: 44
  Present flags
  MAC timestamp: 186447025
  Flags: 0x00
  Data Rate: 6.0 Mb/s
  Channel frequency: 5180 [A 36]
  Channel type: 802.11a (0x0140)
  SSI Signal: -39 dBm
  SSI Noise: -104 dBm
  SSI Signal (Arbitrary Ref): 51 dB
  RX flags: 0x0000
  Vendor namespace: 00:03:7f-0
    Vendor OUI: 00:03:7f
    Vendor sub namespace: 0
    Vendor data length: 10
    .... .... .... 0 0010 0010 110. .... = LSIG-Length: 278
    LSIG: 0x040222cb
    Radio-Type: Cascade (0x00000046)
  [Radiotap Calculated values]
  [QCOM calculated signal: -53 dBm]
```

## 13.7 OCE Overview

This document is the Technical Specification for WFA Optimized Connectivity Experience (OCE). This specification defines architecture, protocols, and functionality for interoperability of Wi-Fi OCE Devices. This section describes the usage of IEEE 802.11 mechanisms required by the OCE program and defines Wi-Fi Alliance specific extensions to the IEEE 802.11 specification (using the vendor specific extension mechanism). The OCE program aims to deliver a better overall connectivity experience by taking advantage of systemic information available within managed networks (for example, hotspot, workplace, operator-deployed networks). The program is intended to address issues identified by operators including very long connection setup times, poor wireless local area network (WLAN) connectivity, and airtime consumed by management frames.

### References

- [1] Wi-Fi Alliance® Wi-Fi Multimedia Technical Specification Version 1.2.0  
<https://www.wi-fi.org/file-member/wmm-including-wmm-power-save-and-admission-control-specification>
- [2] IEEE P802.11ai™/D6.0 Draft Standard for Information Technology –Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment to IEEE P802.11-REVmc™/D6.0: Fast Initial Link Setup

[3] Wi-Fi Alliance® Optimized Connectivity Experience Marketing Task Group, “Marketing Requirements Document for Interoperability Testing of Optimized Connectivity Experience, Version 1.2.1 2016 OCE\_MRD\_v1.2.1.docx

[4] IEEE 802.11™-2012 Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications

[5] Wi-Fi Alliance® Wi-Fi Peer-to-Peer (P2P) Technical Specification Version 1.5

<https://www.wi-fi.org/file-member/wi-fi-peer-to-peer-p2p-technical-specification>

[6] IEEE P802.11-REVmc™/D4.3, Oct 2015 Draft Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications

[7] IETF RFC 6696 “EAP Extensions for the EAP Re-authentication Protocol (ERP)”

[8] Wi-Fi Alliance® Multi Band Operations Technical Specification

## Terms and Definitions

The following definitions and terms are used in this specification.

| Term       | Definition                                    |
|------------|---|
| OCE        | Optimized Connectivity Experience             |
| OCE AP     | An AP that supports OCE features              |
| OCE STA    | A station that supports OCE features          |
| OCE Device | An OCE AP or an OCE STA                       |
| AAA        | Authentication, Authorization, and Accounting |
| AP         | Access Point                                  |
| BSS        | Basic Service Set                             |
| BSSID      | Basic Service Set Identifier                  |
| dB         | Decibels                                      |
| DTIM       | Delivery Traffic Indication Map               |
| EAP        | Extensible Authentication Protocol            |
| EAP-RP     | EAP Re-Authentication Protocol                |
| EDCA       | Enhanced Distributed Channel Access           |
| ESS        | Extended Service Set                          |
| FILS       | Fast Initial Link Setup                       |
| GHz        | Giga Hertz                                    |
| GO         | Group Owner                                   |
| ID         | Identifier                                    |
| IE         | Information Element                           |

| Term     | Definition  |
|----------|---|
| IEEE     | Institute of Electrical and Electronics Engineers |
| LAN      | Local Area Network                                |
| Mbps     | Megabits Per Second                               |
| OOB      | Out Of the Box                                    |
| OUI      | Organizationally Unique Identifier                |
| PMF      | Protected Management Frames                       |
| P2P      | Peer to Peer                                      |
| RADIUS   | Remote Access Dial In User Service                |
| SSID     | Service Set Identifier                            |
| STA      | Client Station                                    |
| STA-CFON | STA Capable of Forming its Own Network*           |
| TBTT     | Target Beacon Transmission Time                   |
| WFA      | Wi-Fi Alliance                                    |
| Wi-Fi™   | Wi-Fi Alliance Trademark                          |

\* An OCE STA-CFON is a STA certified as an OCE STA that is also capable of forming its own network. In addition to operating as an OCE STA, an OCE STA-CFON is capable of operating in a mode where it transmits Beacon frames (for example, a mobile AP, a Managed Wi-Fi Direct GO). An OCE STA vendor must self-declare whether or not the STA is an OCE STA-CFON. The OCE STA-CFON is not required to be WFA certified as a (mobile) AP or Wi-Fi Direct GO.

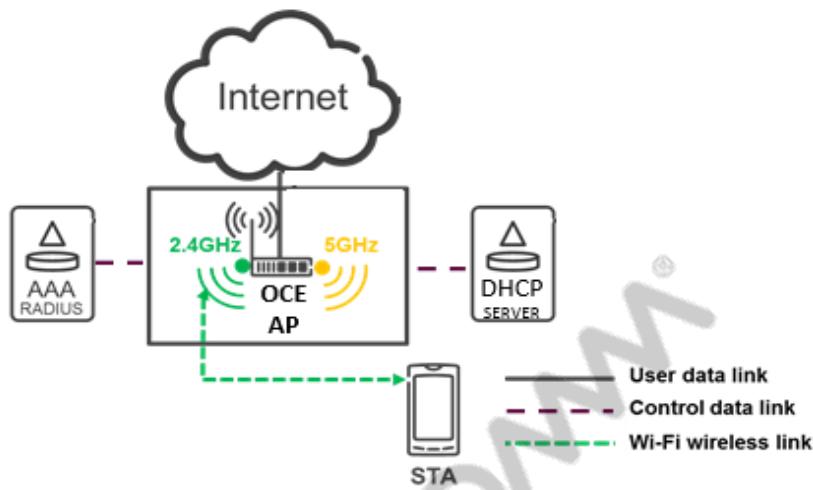
## Optimized Connectivity Experience Architecture

There are multiple components that form an OCE wireless infrastructure network which may vary based on the wireless network deployment. This section describes some of the typical components that form an OCE wireless infrastructure network.

- Access Point (AP): An OCE wireless infrastructure network contains one or more OCE APs.
- WLAN Controller: An OCE wireless infrastructure network contains zero or more WLAN Controllers that may provide centralized management and other features and functions to interconnected AP(s).
- Client Station (STA): An OCE wireless infrastructure network contains zero or more client STAs.
- RADIUS Server: An OCE wireless infrastructure network contains one or more RADIUS Servers that provide Authentication, Authorization, and Accounting (AAA) services.

## Optimized Connectivity Experience Topology

Figure 13-10 shows a typical topology for an OCE infrastructure network. The AAA RADIUS server supports EAP-RP per RFC 6696 in order to authenticate the STA using FILS Shared Key Authentication. The DHCP server can interact with the STA through FILS Higher Layer Setup with Higher Layer Protocol Encapsulation. In some cases, the functionality provided by an AAA RADIUS server and DHCP server may be embedded in the WLAN AP.



**Figure 13-10 Typical OCE Network Topology**

### Optimized Connectivity Experience Specific Requirements

This section specifies the requirements for OCE APs and OCE STAs, which are defined in Section 3:

- OCE Capability Indication
- FILS Discovery Frames
- Reduced Neighbor Report
- Probe Request Frame Transmission Deferral and Suppression
- Indication of non-OCE AP Presence in the operating channel
- Max Channel Time Probe Response Cancellation
- Beacon Frame, or FILS Discovery Frame, replaces a Probe Response frame
- Scanning Mutually Non-Overlapping Channels
- Minimum Beacon Transmission Rate
- Minimum Probe Request Frame Transmission Rate
- Minimum Probe Response Frame Transmission Rate
- RSSI-based Association Rejection Information
- Metrics for AP Selection

In addition, the AAA RADIUS server in an OCE network must support EAP-RP per RFC 6696 [7] in order to authenticate the STA using FILS Shared Key Authentication. This AAA RADIUS server may be discrete or embedded in the OCE AP.

## Required IEEE 802.11 Capabilities

This section specifies additional IEEE 802.11 capabilities, which OCE APs and OCE STAs are required to support.

An OCE AP must support the following capabilities:

- An OCE AP must transmit Probe Response frames to the broadcast address ([2] 10.1.4.3.5) if the request came from an OCE STA and does not contain a FILS Request Parameters element.
- An OCE AP must support FILS Shared Key Authentication ([2] 11.11.2.3.1) per RFC 6696 [7].
- An OCE AP must support FILS Higher Layer Setup ([2] 10.47.3) with Higher Layer Protocol Encapsulation [2] 10.47.3.2].
- An OCE AP must transmit in the FILS Indication element ([2] 8.4.2.178), through Beacon frames and Probe Response frames, a hashed domain name for each of the realms for which it supports FILS Authentication. If more than 7 realms are supported, hashed domain names for any 7 of those realms are transmitted.
- An OCE AP must enable PMF ([4] 4.5.4.9) whenever WPA2 is enabled. An OCE AP with WPA2 enabled must require PMF to be negotiated for use in the association when an OCE STA associates with it.

An OCE STA must support the following capabilities:

- An OCE STA must set the FILSProbeTimer value to 15 ms. The MIB variable dot11FILSProbeDelay is defined in [2] Annex C.3 MIB Detail.
- An OCE STA must support receiving broadcast Probe Response frames ([2] 10.1.4.3.5) and using them in AP selection.
- An OCE STA must support FILS Shared Key Authentication ([2] 11.11.2.3.1) per RFC 6696 [7].
- An OCE STA must support FILS Higher Layer Setup ([2] 10.47.3) with Higher Layer Protocol Encapsulation ([2] 10.47.3.2).
- An OCE STA must support receiving the FILS Indication element ([2] 8.4.2.178) and must use it in its re-authentication decisions.
- An OCE STA must require PMF ([4] 4.5.4.9) to be negotiated for use in the association whenever associating with an OCE AP that has WPA2 enabled.

## OCE Protocol

The goal of the Optimized Connectivity Experience (OCE) certification program is to certify features that deliver a better overall connectivity experience by taking advantage of systemic information available within planned and managed networks (for example, hotspot, workplace, operator-deployed networks). By exchanging this information with each other, APs and STAs enable each other to make intelligent decisions that collectively result in optimization of the connectivity experience.

The following sections describe the information exchanged in this regard and the mechanisms by which APs and STAs exchange the desired information. Where applicable, references to existing

IEEE standards are provided. In instances where there is a need to go beyond what the IEEE standards specify, the corresponding functionality, protocols, and constraints are defined.

## OCE Capability Indication

An OCE AP must transmit an OCE Capability Indication Attribute in the MBO-OCE IE in its Beacon, Probe Response and (Re-)Association Response frames.

An OCE STA must transmit an OCE Capability Indication Attribute in the MBO-OCE IE in its Probe Request and (Re-) Association Request frames.

All OCE devices implementing this version of the OCE specification must set the “OCE Release” sub-field bits of the OCE Control field of the OCE Capability Indication Attribute to 001.

## FILS Discovery Frames

An OCE AP must attempt to transmit either a FILS Discovery frame ([2] 10.47.2, 8.6.8.36) or Beacon frame every 20 TUs. Beacon Interval is a minimum of 100 TUs. In every beacon interval, the OCE AP should attempt to transmit the first FILS Discovery frame 20 TUs after the TBTT.

The exception to the above is if the AP has neighboring OCE APs in the same ESS that are transmitting FILS Discovery frames, and the APs are densely deployed such that STAs in the AP’s coverage area are able to receive FILS Discovery frames from those neighboring APs instead.

An OCE STA-CFON should attempt to transmit either a FILS Discovery frame or Beacon frame every 20 TUs from the time it starts transmitting Beacon frames for a period of at least 3 minutes. In every beacon interval during that period, the OCE STA-CFON should attempt to transmit the first FILS Discovery frame 20 TUs after the TBTT.

The FILS Discovery frame must carry information to indicate the next TBTT of the transmitting AP or STA-CFON ([2] 10.47.2.2).

An OCE AP and STA-CFON must indicate, in the FILS Discovery frame, the primary channel ([2]: 10.47.2.1) of the BSS, when the FILS Discovery frame is transmitted as a non-HT duplicate PPDU. An OCE STA must support receiving a FILS Discovery frame ([2] 10.47.2, 8.6.8.36) and may use it for AP discovery.

## Reduced Neighbor Report

An OCE AP must, when configured to do so, transmit a Reduced Neighbor Report ([2] 10.44.8) in Beacon frames and Probe Response frames, and may transmit a Reduced Neighbor Report in FILS Discovery Frames (Section 3.14).

An OCE AP must transmit the operating channel, TBTT and Short SSID (information field as specified in [2] 8.4.2.169.1) for each of the BSS included in the Reduced Neighbor Report. An OCE AP must include in the Reduced Neighbor Report at least the other BSS that it is currently operating, including BSS it may be operating on other channels/bands, irrespective of ESS.

An OCE STA must support receiving a Reduced Neighbor Report ([2] 10.44.8) in Beacon frames, Probe Response frames and FILS Discovery Frames, and may use the received neighbor information in its scanning algorithms (to prioritize scanning of channels).

## Probe Request Frame Transmission Deferral and Suppression

An OCE STA must, when performing active scan for the purposes of finding an AP with which to (re-)associate, defer transmitting any Probe Request frames until the expiry of the FILSProbeTimer after arriving on the channel, unless it identifies that one or more APs in its current environment are non-OCE APs, or the current environment contains no APs at all based on the STA's most recent scan operation within the past 5 minutes. As an exception, if an OCE STA has pre-configured stealth profiles, it may transmit one directed SSID/BSSID probe per 20 ms for each pre-configured stealth profile on its current channel without waiting for expiry of the FILSProbeTimer.

**NOTE** The means by which an OCE STA determines that one or more APs in its current environment are non-OCE APs is out of scope of OCE certification and is an implementation specific feature.

An OCE STA that has deferred transmitting any Probe Request frames for the duration of the FILSProbeTimer must not subsequently transmit a Probe Request frame while remaining on the channel if it has received a broadcast Probe Response frame, Beacon frame or FILS Discovery frame from an AP that the STA considers a suitable candidate for (re-)association ([2]:10.1.4.3.2.b.2). The OCE STA determines which candidate(s) are suitable based on implementation specific mechanisms.

An OCE STA must transmit a maximum of one Probe Request frame per subsequent 20 ms period on the same channel. As an exception, if an OCE STA has pre-configured stealth profiles, it may transmit one directed SSID/BSSID probe per 20 ms for each pre-configured stealth profile on its current channel without waiting for the subsequent 20 ms period on the same channel.

## Indication of Non-OCE AP Presence in the Operating Channel

An OCE AP must scan its current operating channel to detect the presence of active non-OCE APs. When it identifies any active non-OCE AP in its operating channel, it must set a bit value of 1 in the Non-OCE AP Present sub-field bit of OCE Control field in OCE Capability Indication attribute. Bit B14 of FILS Discovery Frame Control subfield in FILS Discovery frames is reserved according to the IEEE 802.11ai specification. If no active non-OCE APs are detected in its operating channel, or if a valid scan of its operating channel has not yet been completed, an OCE AP must set both sub-field bits to 0.

An OCE STA may use any received Non-OCE AP Present indications in its scanning decisions.

## Max Channel Time Probe Response Cancellation

An OCE STA must transmit the Max Channel Time indication in the FILS Request Parameters element ([2] 8.4.2.173) in a Probe Request frame. The STA must listen for responses for the complete duration indicated in the Max Channel time field of FILS Request Parameter element in its Probe Request frame.

An OCE AP should not transmit a Probe Response frame in response to a Probe Request frame that contains a Max Channel Time indication later than the indicated time period after the reception of the Probe Request frame.

**NOTE** “Should” means that, in normal circumstances, the AP must not transmit a Probe Response frame later than the time indicated by the Max Channel Time indication. However, there may be valid reasons why, in particular circumstances such as when the AP is unable to gain access to the channel in a congested environment, the AP transmits a Probe Response frame later than the time indicated by the Max Channel Time indication.

### **Beacon frame, or FILS Discovery Frame, replaces a Probe Response frame**

An OCE AP must not transmit a Probe Response frame in response to a Probe Request frame containing a Max Channel Time indication if a Beacon frame ([2] 10.1.4.3.4) or FILS Discovery frame is scheduled to be transmitted within the time period that is indicated in the Max Channel Time field in the Probe Request frame.

An exception to the above rule is when the Probe Request frame contains a FILS Request Parameters element ([2] 8.4.2.173) containing criteria not present in Beacon or FILS Discovery Frames that the AP matches.

### **Scanning Mutually Non-Overlapping channels**

When performing scanning in the 2.4 GHz band for the purpose of finding an AP with which to (re-)associate, an OCE STA must first scan on channels 1, 6 and 11, by active or passive scanning procedures, before transmitting Probe Request frames on any other 2.4 GHz channel (if required) unless it expects to find a desired AP on a different channel (for example, in case the STA has cached the AP’s information from having been previously associated to it).

An OCE AP (not including OCE STA-CFONs) must select, OOB, one of channels 1, 6 or 11 when operating in the 2.4 GHz band. The method used by the AP to select one of those channels is out-of-scope of this specification.

### **Minimum Beacon Transmission Rate**

An OCE AP must transmit Beacon frames at a minimum rate of 5.5 Mbps on the 2.4 GHz band, OOB. An OCE STA-CFON must transmit Beacon frames at a minimum rate of 5.5 Mbps on the 2.4 GHz band, OOB.

### **Minimum Probe Request Frame Transmission Rate**

An OCE STA must, except as indicated below, transmit Probe Request frames at a minimum rate of 5.5 Mbps on the 2.4 GHz band OOB.

The exception to the above is when the OCE STA is performing active scan for the purpose of finding an AP with which to (re-)associate, and identifies that one or more APs in its current environment are non-OCE APs. In this case, the OCE STA must transmit the first Probe Request frame after arriving on the channel at a minimum rate of 5.5 Mbps on the 2.4 GHz band OOB, but can transmit subsequent Probe Request frames at any rate while remaining on the channel.

**NOTE** The means by which an OCE STA determines that one or more APs in its current environment are non-OCE APs is out of scope of OCE specification and is an implementation specific feature.

## Minimum Probe Response Frame Transmission Rate

An OCE AP must transmit Probe Response frames at a minimum rate of 5.5 Mbps on the 2.4 GHz band OOB, except for (a) unicast responses to Probe Request frames from non-OCE STAs which can be transmitted at any rate allowed as per IEEE 802.11-2012 [4], and (b) unicast responses to Probe Request frames from OCE STAs received at a rate below 5.5 Mbps which can be transmitted at the same rate as the Probe Request frame.

An OCE STA-CFON must transmit Probe Response frames at a minimum rate of 5.5 Mbps on the 2.4 GHz band OOB, except for unicast responses to Probe Request frames from non-OCE STAs which can be transmitted at any rate allowed as per IEEE 802.11-2012 [4].

## RSSI-based Association Rejection Information

An OCE AP must, when configured to do so, reject a (Re-)Association request if the RSSI of the (Re-)Association request received from a STA is below a minimum threshold value. This minimum threshold value is within the range of -60 to -90 dBm. How an OCE AP selects the minimum threshold value and field deployment configurability of an OCE AP with the minimum threshold value are out of scope of this specification.

An OCE AP that rejects a (re-)association request on the basis of insufficient RSSI must send a (Re-)Association Response frame to the requesting STA that:

- Must have the Status Code field set to the value 34
- Must include the RSSI-based Association Rejection attribute in the MBO-OCE Information Element (MBO-OCE IE) with fields set as follows:
  - Delta RSSI set to a positive value in dB indicating the difference between the minimum RSSI at which the AP would accept a (re-)association request from the requesting STA and the AP's measurement of the RSSI at which the (re-) association request was received
  - Retry Delay set to a non-negative value in seconds for which the AP will not accept any subsequent (re-)association requests from the requesting STA, irrespective of the RSSI at which they may be received.

When an OCE STA receives a (Re-)Association Response frame from an AP containing the RSSI-based Association Rejection attribute in the MBO-OCE IE, that STA must not send a (Re-)Association Request frame or a unicast Probe Request frame to that AP until both of the following criteria are met:

- The time period indicated in the Retry Delay field of the most recently received RSSI-based Association Rejection attribute has expired, AND
- Based on the value indicated in the Delta RSSI field of the most recently received RSSI-based Association Rejection attribute, the STA estimates that the AP will receive its (Re-)Association Request frame or unicast Probe Request frame with at least the minimum sufficient RSSI.

## Metrics for AP Selection

An OCE AP must transmit the BSS Load element ([4] 8.4.2.30) in its Beacon and Probe Response frames. An OCE AP that supports VHT PHY may also transmit the Extended BSS Load element ([6] 8.4.2.159) in its Beacon and Probe Response frames.

An OCE AP must transmit Estimated Service Parameters (ESP) element ([6] 8.4.2.171) in its Beacon and Probe Response frames. An OCE AP must include ESP information for AC\_BE in the Estimated Service Parameters element and may include ESP information for other access categories.

An OCE AP that has an estimate of the available WAN backhaul link capacity must transmit the Reduced WAN Metrics attribute of the MBO-OCE IE in its Beacon Frames and Probe Response Frames. If estimates of the available WAN backhaul link capacity are unavailable, the OCE AP need not transmit this attribute. The current available (unused) WAN backhaul capacity is the product of the WAN backhaul link current speed and one minus the current fractional loading of the WAN backhaul link. The algorithm to calculate currently available (unused) WAN backhaul link capacity is implementation dependent.

### **MBO-OCE IE, Optimized Connectivity Experience Attributes, and Frame Formats**

This section defines the OCE Attributes in the MBO-OCE IE, and frame formats for OCE APs and STAs and the OCE communication protocol.

#### **MBO-OCE IE**

The Vendor Specific information element format is used to define the MBO-OCE IE in this specification. The format of the MBO-OCE IE is defined in MBO Technical Specification [8] and described in the following table. Attribute IDs 101-150 are reserved for OCE.

**Table 13-1 OCE IE Format**

| Field                  | Size (Octets) | Value (Hex) | Description   |
|------------------------|---------------|-------------|---|
| Element ID             | 1             | 0xDD        | IEEE 802.11 vendor specific information element   |
| Length                 | 1             | Variable    | Length of the following fields in the IE in octets. The Length field is a variable, and set to 4 plus the total length of the MBO and OCE Attributes. |
| OUI                    | 3             | 0x50-6F-9A  | WFA specific OUI (refer to sub-clause 8.4.1.31 of [6]).   |
| OUI Type               | 1             | 0x16        | Identifying the type and version of the MBO-OCE IE  |
| MBO and OCE Attributes | Variable      | Variable    | One or more MBO and/or OCE Attribute(s)   |

#### **OCE Attributes**

The OCE Attributes are defined to have a common general format consisting of a one (1) octet OCE Attribute ID field, a one (1) octet Length field, and variable-length attribute-specific Information fields, as shown in [Table 13-2](#).

**Table 13-2 OCE Attribute General Format**

| Field                | Size (Octets) | Value (Hex) | Description                                     |
|----------------------|---------------|-------------|---|
| Attribute ID         | 1             | Variable    | Identifies the type of OCE Attribute.           |
| Length               | 1             | Variable    | Length of the following fields in the attribute |
| Attribute Body Field | Variable      | Variable    | OCE Attribute specific information fields       |

The following table defines the OCE Attributes and whether their support is mandatory or optional. The OCE Attributes are inserted in the IEEE 802.11 management frames used for OCE purposes as specified in [Table 13-3](#).

**Table 13-3 OCE Attributes**

| Attribute ID | Field Description                     | Management Frame Type                                |  | Mandatory/Optional |     |
|--------------|---------------------------------------|--|--|--------------------|-----|
|              |                                       | AP   | STA  | AP                 | STA |
| 101          | OCE Capability Indication             | Beacon, Probe Response and (Re-)Association Response | Probe Request and (Re-)Association Request | M                  | M   |
| 102          | RSSI-based (Re-)Association Rejection | (Re-) Association Response                           | NA   | M                  | M   |
| 103          | Reduced WAN Metrics                   | Beacon and Probe Response                            | NA   | M                  | M   |
| 104-150      | Reserved                              | NA   | NA   | NA                 | NA  |

**NOTE** Mandatory/optional indication in the table refers to implementation of the feature. The indication does not signify whether the device always transmits the attribute in the corresponding frames.

### OCE Capability Indication Attribute

The format of the OCE Capability Indication Attribute is illustrated in [Table 13-4](#).

The OCE Capability Indication Attribute is transmitted by an OCE AP in Beacon, Probe Response and (Re-)Association Response frames and by an OCE STA in Probe Request and (Re-) Association Request frames. This attribute is mandatory to transmit by all OCE devices and indicates the release of OCE that the device supports. The release of OCE documented in this specification is release 1 and OCE devices implementing this specification will set the “OCE Release” field to 0x01.

**Table 13-4 OCE Capability Indication Attribute**

| Field Name        | Size (Octets) | Value              | Description  |
|-------------------|---------------|--------------------|--|
| Attribute ID      | 1             | 0x65 (decimal 101) | Identifies the type of OCE attribute.  |
| Length            | 1             | 0x01               | Length of the following fields in the attribute.   |
| OCE Control field | 1             | Variable           | Carries information on latest release of OCE that the device supports, along with other indicators (see <a href="#">Table 13-5</a> and <a href="#">Table 13-6</a> ). |

**Table 13-5 OCE Control field as advertised by an OCE AP**

|  |       |    |         |
|--|-------|----|---------|
|  | B0 B2 | B3 | B4 - B7 |
|--|-------|----|---------|

**Table 13-5 OCE Control field as advertised by an OCE AP**

|      | OCE Release | Non-OCE AP Present | Reserved |
|------|-------------|--------------------|----------|
| Bits | 3           | 1                  | 4        |

**Table 13-6 OCE Control field as advertised by an OCE STA**

|      | B0 - B2     | B3 - B7  |
|------|-------------|----------|
|      | OCE Release | Reserved |
| Bits | 3           | 5        |

### RSSI-based (Re-)Association Rejection Attribute

The format of the OCE RSSI-based (Re-) Association Rejection attribute is illustrated in [Table 13-7](#).

The OCE RSSI-based Association Rejection attribute is provided by an OCE AP in a (Re-) Association Response frame to indicate that the AP has rejected the STA's (re-)association request on the basis of insufficient RSSI. This attribute is mandatory in the (Re-)Association Response frame if the reason for the (re-)association rejection is insufficient RSSI (i.e. Status Code field value = 34).

The value of the Delta RSSI field is an 8-bit integer between 0 and 127, equal to the difference in dB between the minimum RSSI at which the AP would accept a (re-)association request from the requesting STA and the AP's measurement of the RSSI at which the (re-) association request was received.

The value of the Retry Delay field is an 8-bit integer between 0 and 255, equal to the time period in seconds for which the AP will not accept any subsequent (re-)association requests from the requesting STA.

**Table 13-7 RSSI-based Association Rejection Attribute**

| Field Name   | Size (Octets) | Value              | Description   |
|--------------|---------------|--------------------|---|
| Attribute ID | 1             | 0x66 (decimal 102) | Identifies the type of OCE attribute.   |
| Length       | 1             | 0x02               | Length of the following fields in the attribute.  |
| Delta RSSI   | 1             | Variable           | The difference in dB between the minimum RSSI at which the AP would accept a (re-)association request from the requesting STA and the AP's measurement of the RSSI at which the (re-)association request was received |
| Retry Delay  | 1             | Variable           | The time period in seconds for which the AP will not accept any subsequent (re-)association requests from the requesting STA  |

### Reduced WAN Metrics Attribute

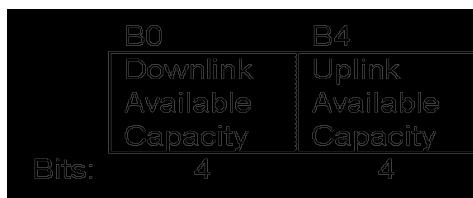
The format of the OCE Reduced WAN Metrics Attribute is illustrated in [Table 13-8](#).

The OCE Reduced WAN Metrics Attribute is transmitted by an OCE AP in Beacon and Probe Response frames to aid an OCE STA in AP selection. This attribute is mandatory to include in all Beacon and Probe Response frames if the OCE AP has an estimate of the available WAN backhaul link capacity.

**Table 13-8 Reduced WAN Metrics Attribute**

| Field Name         | Size (Octets) | Value              | Description                                      |
|--------------------|---------------|--------------------|--|
| Attribute ID       | 1             | 0x67 (decimal 103) | Identifies the type of OCE attribute.            |
| Length             | 1             | 0x01               | Length of the following fields in the attribute. |
| Available Capacity | 1             | 1                  | See definition below                             |

The Available Capacity field is a bit-mapped field with the 4 MSB representing the Downlink Available Capacity and the 4 LSB representing the Uplink Available Capacity, as defined in [Figure 13-11](#).

**Figure 13-11 Available Capacity field in Reduced WAN Metrics Attribute**

- Downlink Available Capacity: A 4-bit integer N representing an estimate of the currently available (unused) WAN backhaul link capacity in the downlink, according to the following formula.

- Uplink Available Capacity: A 4-bit integer N representing an estimate of the currently available (unused) WAN backhaul link capacity in the uplink, according to the following formula.

N is selected as the smallest integer in the range 0 – 15 obeying the following inequality, where the available WAN backhaul link capacity is  $C_{DL/UL}$ :

$$C_{DL/UL} \leq 2^N \times 100 \text{ kbit/s}$$

N is set to 15 when  $C_{DL/UL}$  is greater than 3,276,800 kbit/sec.

## 13.8 HotSpot 2.0

HotSpot 2.0 functionality in an AP consists of advertising a set of features, providing information to prospective clients, and offering services to associated clients. Critical information is available in Beacon/Probe Response frames as Information Elements, while non-critical information may be queried using the GAS/ANQP framework. Services offered include Proxy ARP, L2TIF, and DGAF Disable.

### 13.8.1 Beacon/Probe Response IE Processing

#### 13.8.1.1 Extended Capabilities IE

The Interworking bit and the TDLS Prohibited bit in the Extended Capabilities IE is set by hostapd (in the function `hostapd_eid_ext_capab`), passed to the driver, and processed in `wlan_mlme_parse_appie`. These bits are stored in `iv_hotspot_xcaps`. The Proxy ARP bit is set in this IE in `ieee80211_add_extcap`.

#### 13.8.1.2 Interworking IE

The Interworking IE contains Access Network Options and optionally Venue Info and HESSID. These are configured in hostapd (in the function `hostapd_eid_interworking`), passed to the driver, and processed in `wlan_mlme_parse_appie`. Presence of Interworking IE indicates support for the GAS (Generic Advertisement Service) protocol.

#### 13.8.1.3 Advertising Protocol IE

The Advertising Protocol IE indicates support from ANQP and is passed to the driver by hostapd (in the function `hostapd_eid_adv_proto`) to be included in Beacon and Probe Responses.

#### 13.8.1.4 Roaming Consortium IE

The Roaming Consortium IE is configured in hostapd (in the function `hostapd_eid_roaming_consortium`) and passed to the driver to be included in Beacon and Probe Responses.

## 13.8.2 Probe Request Processing

When configured as a Hotspot 2.0 compliant AP, probe requests from clients that do not support Interworking would elicit a probe response from the AP, but clients that support Interworking will not elicit a probe response unless the parameters in the Interworking element match with those on the AP. Clients that support Interworking will have an Interworking IE and Interworking bit set in the Extended Capabilities IE. The HESSID in the Interworking IE, if present, must match the HESSID of the AP or must be a Broadcast Address. Furthermore, the Access Network Options in the Interworking IE must match those of the AP. A probe response is transmitted if these conditions are met. This processing is performed in ieee80211\_recv\_probereq.

## 13.8.3 Action Frame Processing

### 13.8.3.1 DLS Request

DLS Requests are turned down in ieee80211\_recv\_action by sending a DLS response with a status code signifying that DLS is not allowed in the BSS by policy.

### 13.8.3.2 GAS/ANQP Processing

The GAS protocol is implemented in hostapd, as is ANQP. Public action frames are sent to hostapd as wireless events with the tag “Manage.action”, and these events are processed in hostapd in the function atheros\_wireless\_event\_wireless\_custom. The GAS/ANQP processing is carried out in the file gas.c in hostapd.

#### ANQP IEs Configured in Hostapd

When ANQP IEs are configured in hostapd, hostapd responds with ANQP IEs in one or more GAS Response frames.

#### ANQP IEs Configured in AnqpServer

When ANQP IEs are configured in an external AnqpServer, hostapd forwards the ANQP queries it receives to the AnqpServer and sends a GAS Initial Response to the client with a suitable “Comeback Delay”. The Comeback Delay is previously negotiated between hostapd and AnqpServer for each ANQP IE. The AnqpServer formats a response to the ANQP query and sends it to the hostapd, so that when hostapd receives a GAS Comeback Request from the client, it responds with one or more GAS Comeback Responses. If the entire payload cannot be sent in one GAS Comeback Response, it is split over several GAS Comeback Responses.

## 13.8.4 Proxy ARP

When this feature is enabled, the bit for Proxy ARP in the Extended Capabilities IE in Beacon/Probe Response frames is set to 1.

Proxy ARP is implemented in the file drivers/wlan/os/linux/src/osif\_proxyarp.c. When this feature is enabled, the function wlan\_proxy\_arp is called for every frame being transmitted on the

WLAN. This function checks to see if the frame to be transmitted is one of the following and if it is, an appropriate action is performed.

- ARP Request – Respond with ARP Response, and discard frame transmission.
- Gratuitous ARP Request – Learn IP-MAC address mapping, and discard frame transmission.
- ARP Reply or Gratuitous ARP Reply – Discard frame
- DHCP Ack – Learn IP-MAC address mapping, and let frame be transmitted to associated client.
- Neighbor Solicitation – If this is a neighbor solicitation for Duplicate Address Detection, the frame is allowed to be transmitted. If it is a neighbor solicitation from an associated client, IPv6-MAC address mapping is learnt. In all other cases, respond with Neighbor Advertisement, and discard frame transmission.
- Neighbor Advertisement – Discard unsolicited neighbor advertisements, otherwise let frame be transmitted to associated client.

### 13.8.5 DGAF Disable

When DGAF (Downstream Group Address Forwarding) is disabled, the DGAF Disabled bit in the HS2.0 IE in Beacon/Probe Response frames is set to 1.

This feature is dependent on Proxy ARP and is enabled only if Proxy ARP is also enabled. Broadcast DHCP Offer and DHCP Ack frames are converted to unicast frames and transmitted to the associated client. All other broadcast/multicast frames are discarded and not transmitted to associated clients.

### 13.8.6 L2TIF

L2TIF (Layer 2 Traffic Isolation and Filtering) can be implemented in two flavors – AP mode or Firewall mode. The AP mode requires that all frames from a client be forwarded to a Portal that enforces the traffic policy by either discarding frames or forwarding them to an eventual destination. The Firewall mode requires that all frames be acted upon the applicable firewall rules. The current implementation enables AP mode. The Firewall mode can be enabled by integrating NetFilter, ebtables, iptables into the solution. (More information on NetFilter and iptables can be found at <http://www.netfilter.org/>)

When L2TIF is enabled, intra-BSS forwarding is prevented and all frames from a client (even those that are destined to another client on the same BSS) are forwarded to the Ethernet portal. This is done by forwarding the frames to the local network stack. The relevant code can be seen in the function `ieee80211_deliver_data`.

The Linux kernel bridge is a “learning” bridge and keeps track of the MAC addresses located on its ports. Hence, frames from one client to another client in the same BSS will not be forwarded to the Ethernet portal by the Linux kernel bridge as both the source and destination are on the same port of the bridge. The Linux kernel bridge is modified to enable all frames from a client to be forwarded to the Ethernet portal irrespective of the destination when L2TIF is enabled.

## 13.9 Hidden SSID support in strict passive scan

| Feature                                    | IPQ4019.ILQ.4.0 |             |             |             | IPQ8064.ILQ.4.0 |             |    |             |             | QCA9531.ILQ.4.0 |             |             |             | QCA9558.ILQ.4.0 |             |             | QCA9563.ILQ.4.0 |             |             |             |             |   |
|--|-----------------|-------------|-------------|-------------|-----------------|-------------|----|-------------|-------------|-----------------|-------------|-------------|-------------|-----------------|-------------|-------------|-----------------|-------------|-------------|-------------|-------------|---|
|  | IPQ<br>4019     | QCA<br>9886 | QCA<br>9984 | QCA<br>9889 | QCA<br>9984     | QCA<br>9980 | AR | QCA<br>9380 | QCA<br>9880 | QCA<br>9889     | QCA<br>9880 | QCA<br>9889 | QCA<br>9531 | QCA<br>9886     | QCA<br>9984 | QCA<br>9558 | QCA<br>9880     | QCA<br>9889 | QCA<br>9563 | QCA<br>9880 | QCA<br>9886 |   |
|  | 3.14            |             |             |             | 3.14            |             |    |             |             | 3.3.8           |             |             |             |                 |             |             |                 |             |             |             |             |   |
| Hidden SSID support in strict passive scan | ✓               | ✓           | ✓           | ✓           | ✓               | ✓           |    |             | ✓           | ✓               | ✓           | ✓           | ✓           | ✓               | ✓           | ✓           | ✓               | ✓           | ✓           | ✓           | ✓           | ✓ |

In general, APs broadcast beacons periodically and in the beacon SSID string is present. However, to provide security, sometimes APs do not give the SSID information in the beacon. It is known as hidden SSID. In case the client that wants to connect to an AP, already knows the SSID can connect to the AP by sending a directed probe request to the AP and getting a probe response.

Therefore, for a client to be able to connect to a hidden SSID AP, the client needs to perform an active scan (sending probe in the channel where the AP is present). However, there are channels which are passive channels (where one can send a probe request only after seeing a beacon in that channel). Therefore, even if an AP is in a passive channel, connection to it is possible.

Recently, some customer have requested strict passive scan where the rule is not to send any probe requests in that channel at all even after seeing beacon. This strict passive scan is used, perhaps, to reduce power consumption and/or to reduce the traffic in the BSS or some other regulatory restrictions. Now, when strict passive scan is used (no probe requests at all), connection to a hidden SSID AP is not possible using the traditional method of candidate AP selection. The traditional method of selection collects beacons or probe responses and add them in a list and then select the AP with the appropriate SSID.

However, in case of strict passive scan, although the beacon from the AP is present, the SSID is not available; therefore, it is not possible to choose the candidate AP to which the station must connect. The station is unable to connect to hidden-SSID AP with strict passive scan.

### Implementation

To overcome the hidden SSID issue with strict passive scan come up with new solution. Because the beacon information from the hidden SSID AP is available, all the information about the AP is available, except the SSID. Also, it is possible that there are more than one hidden SSID APs. A list of all the beacons/probe-response from various APs are available and the AP to which the connection must be made is not clear at this stage.

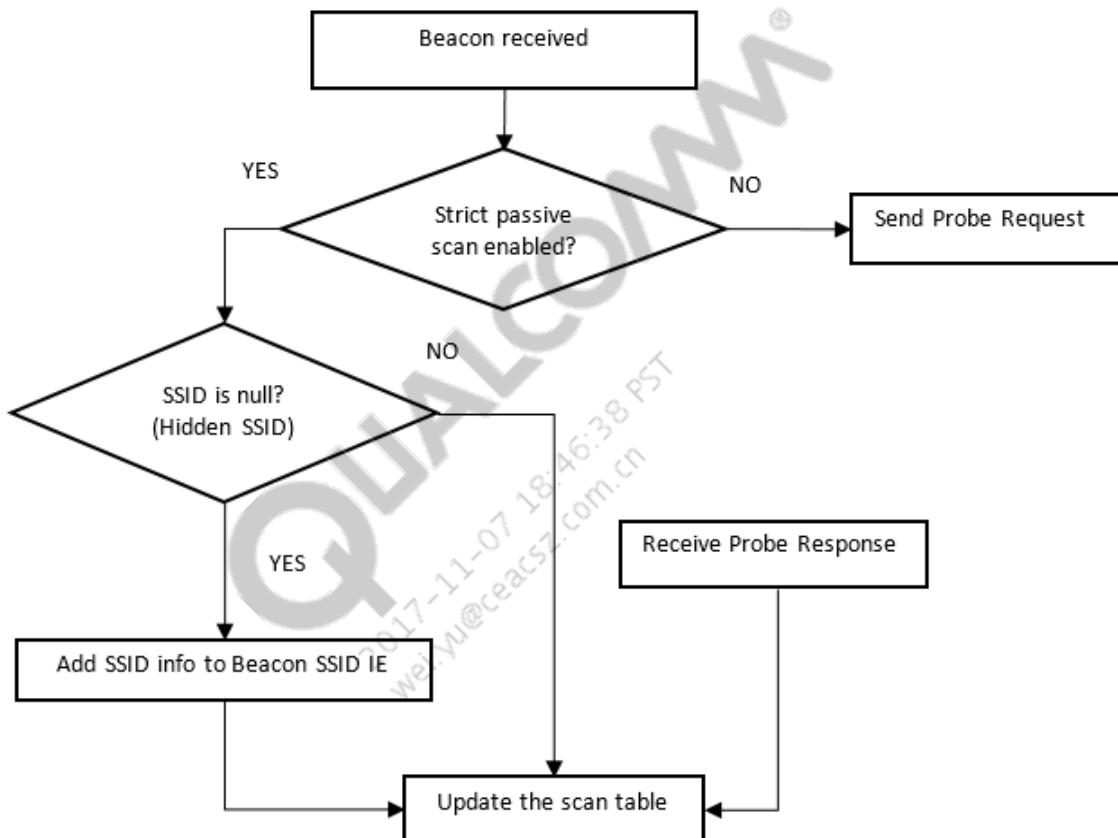
This problem can be resolved by adding one extra piece of information in the client software. The extra piece of information is the BSSID. Therefore, instead of setting SSID only, a SSID and BSSID must be set.

In general, a user must be able to determine and configure SSID to connect to AP sending beacon with hidden SSID. Similar to that in the strict passive scan case, for the client/STA to connect to AP with hidden SSID, a user must know the BSSID of the hidden AP to connect along with the hidden AP's SSID.

When a user configures the hidden AP SSID during the STA configuration, it is also necessary to configure the BSSID of hidden AP.

When Client/STA receive beacon from hidden SSID AP, it check is the beacon is received from the configured BSSID, if so, it updates the SSID info in the Beacon, in turn it updates the SCAN TABLE. From this user get the BSSID, SSID info in the scan entry.

Then STA can select the AP, and Authenticate to Selected AP. As in strict passive channel no Probe Request can be sent. So Client/STA does not get the Probe Response and thus no SSID information in the SCAN TABLE. With this implementation without probe request and response, Client/STA able to authenticate as Scan Entry updated to configured BSSID and SSID



**Figure 13-12 Flow of updating SSID info for beacon received with hidden SSID**

## Commands

The following commands are used to set/get the BSSID of hidden SSID AP for which Client/STA needs to connect.

```
iwpriv athx conf_bssid 'bssid of the hidden AP'  
iwpriv athx get_conf_bssid
```

Example: iwpriv athx conf\_bssid 8C:FD:F0:06:B2:2F

**NOTE** To test this behavior, strict passive scan must also be enabled.

To enable strict passive scan, use the following command:

```
iwpriv wifix pas_scanen 1
```

## 13.10 Multiband Operation (MBO) overview

This section describes the design and features of MBO protocol implemented in the code base of the access point. MBO facilitates efficient use of multiple frequency bands and channels that are available to access points (APs), and the wireless devices that associate with them (stations). MBO offers solutions for band steering, load balancing, and other related operational procedures. The features defined for MBO are band-independent and apply across all available unlicensed bands where Wi-Fi systems operate. With exchange of information between APs and STAs, intelligent decisions are made to collectively drive the network towards more efficient use of the available spectrum.

The following table describes all the terms and acronyms used in the description of MBO functionalities:

| Term                     | Definition   |
|--------------------------|--|
| MBO IE                   | Multiband Operation Information Element  |
| IE                       | Information Element  |
| BTM                      | BSS Transition Management  |
| Req                      | Request frames   |
| OUI                      | Organizationally Unique Identifier   |
| STA                      | Station  |
| Candidate AP             | An AP with which the STA is currently not associated.  |
| MBO AP                   | An AP with MBO features and functions enabled.   |
| MBO STA                  | A station with MBO features and functions enabled.   |
| Multimode STA            | A station that contains both Wi-Fi and cellular capabilities                                       |
| MBO multimode STA        | A station with MBO features and functions enabled, in addition to Wi-Fi and cellular capabilities. |
| Serving AP               | The AP to which an STA is associated   |
| Wi-Fi MBO Capable Device | An MBO AP or an MBO STA  |

The implementation of MBO features are contained within the MBO module of the UMAC layer (UMAC/MBO). The ieee80211\_mbo\_priv.h file within the MBO module also contains the data structures of the IEs and the MBO data structure needed to be included within the VAP code so that all MBO related parameters can be stored within the VAP and used as and when required. The ieee80211\_mbo.c file contains all MBO related functions which broadly falls into the categories of storage of values set by user, returning values sent by user to the application layer, adding MBO related IEs to the end of frames.

### MBO IE overview

The MBO IE is included in the beacon, probe response, reassociation response, and BTM request frames. In the function that is used to construct these frames, a condition is included to obtain the pointer at the end of the frame and add to it the MBO IE only if MBO is enabled previously using an IOCTL constructed for this purpose. Two IOCTLs are constructed for this purpose: one IOCTL is used to enable or disable MBO depending on whether the value passed is 0 or 1 respectively, and

the other IOCTL is used to check whether the MBO is enabled or disabled in its present state. After the MBO is enabled or disabled using the IOCTL the information is passed by the operating system to the UMAC layer. The information is further transmitted to the MBO modules so that a flag can be set (if MBO is enabled) or cleared (if MBO is disabled). This flag is stored inside the MBO data structure contained within the VAP and can be referenced whenever required. The UMAC layer returns the flag value to the OS if the IOCTL to determine the flag status is configured.

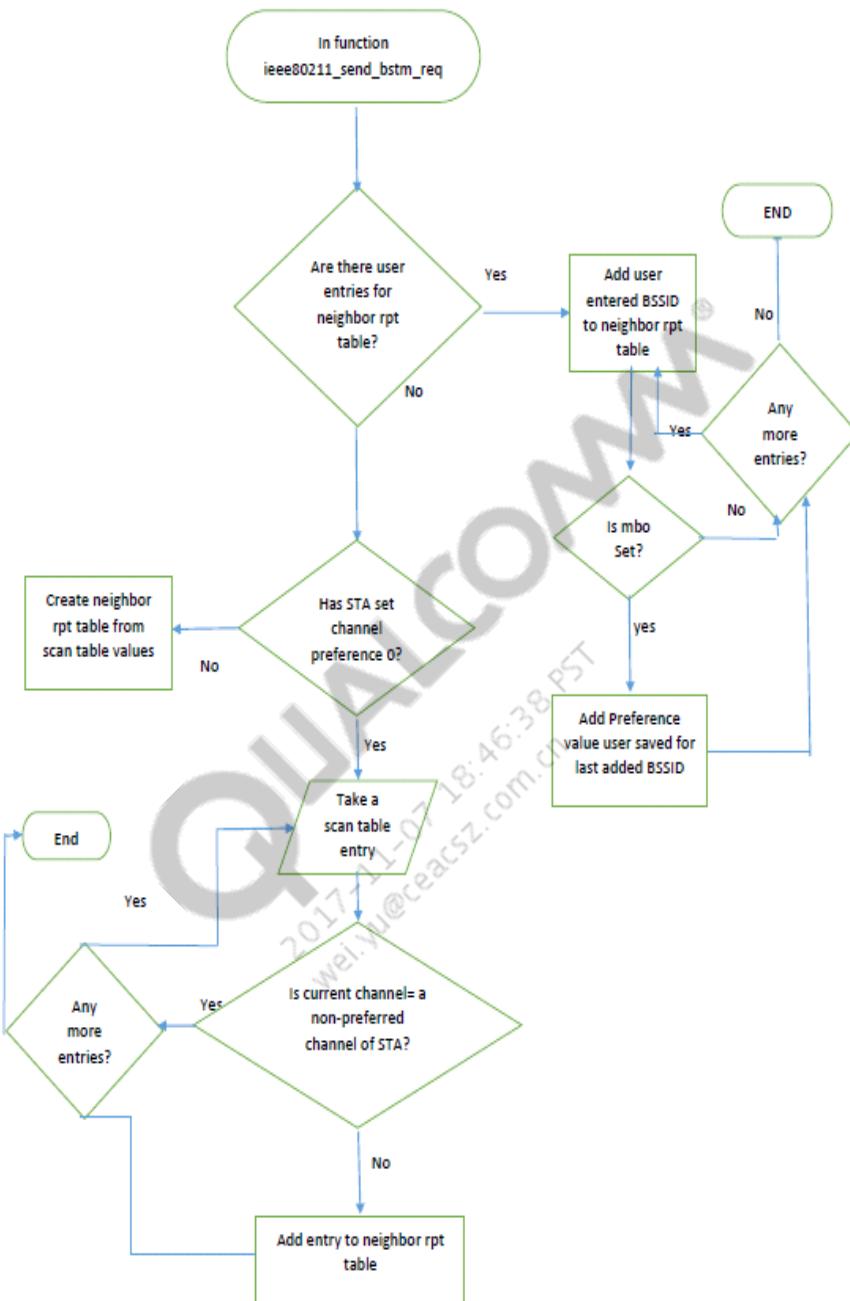
### **MBO Capability Attribute and Association Disallowed Attribute IEs overview**

The data structure for the MBO Capability Attribute and Association Disallowed Attribute is included in the `ieee80211_mbo_priv.h` file of the MBO module. Separate IOCTLs are used to set values for these two attributes. If zero is passed as a value to both the IOCTLs, both the attributes are disabled. After the user configures a particular IOCTL, the information passes from the operating system to the UMAC layer where the value is stored within the MBO data structure contained in the VAP's internal storage. When the bytes of the beacon frames, probe response frames, and re-association frames are attached to the frames, the function to attach MBO IE is called, only if MBO is enabled. The same function also checks whether the user has set values for MBO Capability Attribute and Association Disallowed Attribute IEs and attaches the related IEs if they are enabled.

Different IOCTLs exist to retrieve the value set for these attributes by the user. When the user configures such an IOCTL, the information stored internally in the VAP is passed by the UMAC to the operating system and finally the application layer where it is displayed.

### **Configure the neighbor report table**

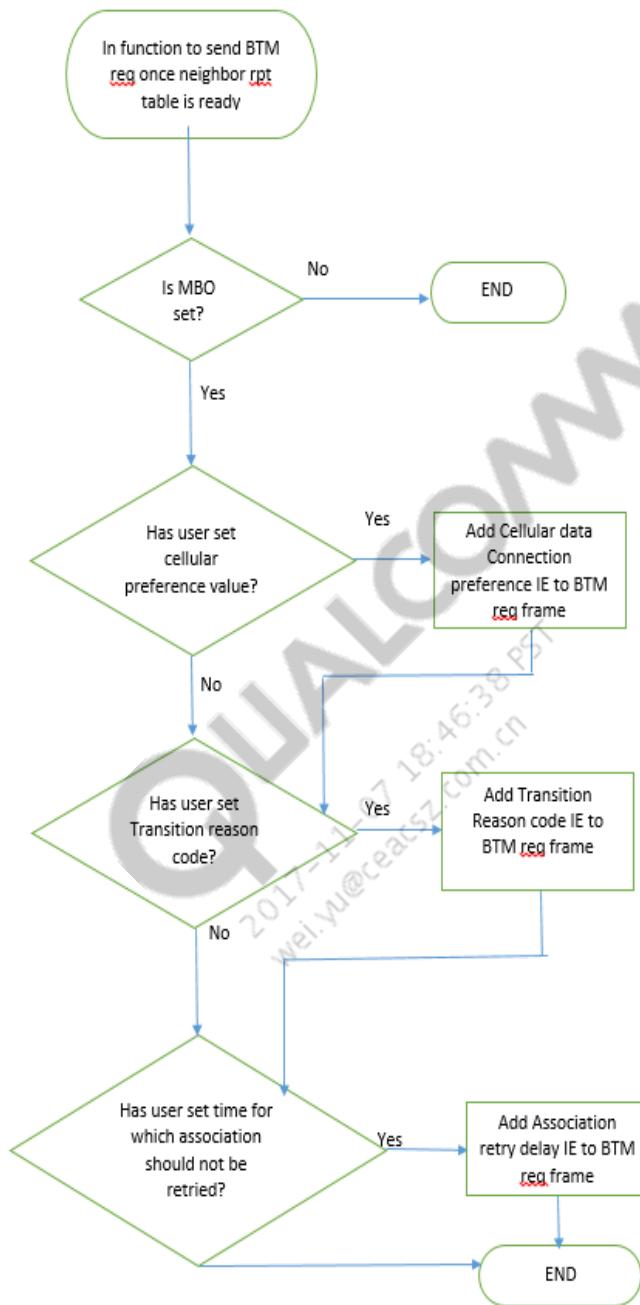
An IOCTL is used to set values for the neighbor report table. The values set by the user are stored within a structure inside the Wireless Network Management structure within the VAP. When the user enters a MAC address-preference ID pair, it is saved in the VAP's internal storage. Up to 32 pairs can be stored in this manner. The stored values can be changed by modifying the `MAX_BSSID_STORE` macro defined inside the `ieee80211_wnm.h` file. If the user passes zero as a parameter to the same IOCTL, the older values are cleared. When the user sets values for the neighbor report table, a flag is set within the VAP's internal storage. This flag is also used to track the number of pairs that a user enters. If this flag has a non-zero value, the user-specified values are copied to the neighbor report table to be included in the BTM Request frames. If user does not use the IOCTL to set neighbor report table values or clears values contained in it, the flag is reset. In this case, the scan table values are used to create the neighbor report table.



**Figure 13-13 Work flow of neighbor report table configuration**

### Inclusion of IEs in BTM request frames

The data structures of the Cellular Data Connection Preference attribute IE, the Transition Reason Code attribute IE, and the Association Retry Delay attribute IE are placed inside the `ieee80211_mbo_priv.h` header file of the MBO module. Instead of using different IOCTLs for each IE, they can be specified using the same IOCTL that is used to send unsolicited BTM request frames. The specified values are stored within the `bstmreq` structure of type `ieee80211_bstm_reqinfo`. This `ieee80211_bstm_reqinfo` data structure stores other parameters related to BTM Request frames. Separate IOCTLs are present to retrieve the set values as needed.



**Figure 13-14 Work flow of inclusion of IEs to BTM request frames**

### Channel and Band Indication and Preference

An MBO STA informs the serving MBO AP about the channels in which it does not operate, or decides to not operate through the re-association request frames. If this information that the MBO STA sends to the serving MBO AP changes while the STA is still associated with the AP, the new information is communicated through WNM-notification request frame. Assuming the MBO STA functions according to the specifications, the MBO AP functions in the following manner:

- MBO AP stores the non-preferred channels for this MBO STA.
- MBO AP utilizes the information supplied in the Non-preferred Channel Report Attributes for associated MBO STAs as an input when requesting a BSS transition for that specific MBO STA.
- MBO AP filters out all those APs from the neighbor report table of BTM request, that are currently operating on channels indicated as non-preferred by the recipient STA of that BTM request frame.

## 13.11 Wi-Fi Alliance MBO functions

The Technical Specification for WFA Multiband Operation (MBO) specification defines architecture, protocols, and functionality for interoperability of Wi-Fi MBO Capable Devices. This specification describes IEEE 802.11 mechanisms used in the MBO program and defines vendor specific extensions to the IEEE 802.11 specification. These extensions facilitate efficient use of multiple frequency bands available to Access Points and the wireless user devices that may associate with them.

This specification provides practical solutions for band steering, load balancing, and other related operational procedures. The extensions defined in this specification are band independent and are expected to be usable across all available unlicensed bands where Wi-Fi systems can operate.

### References

For more information about the MBO protocol, refer to the following resources:

- Wi-Fi Alliance® Wi-Fi
- Multiband Operation Marketing Task Group, “Marketing Requirements Document for Interoperability Testing of Wi-Fi Multiband Operations, Version 1.01”, 2014
- Wi-Fi Alliance® Wi-Fi Multiband Operation Marketing Task Group, “Specification Requirements Document for Wi-Fi Multiband Operations Version 1.01,” 2014
- IEEE Computer Society, “IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,” (IEEE Std. 802.11mc/D8.0), January 2015

### Terms and Acronyms

The following definitions and terms are used in this specification.

**Table 13-9 Terms and definitions for Wi-Fi multiband operation**

| Term         | Definition   |
|--------------|--|
| Candidate AP | An AP to which the STA is currently not associated |
| MBO          | Multiband Operation                                |
| MBO AP       | An AP that has MBO features and functions enabled  |

|                          |   |
|--------------------------|---|
| MBO STA                  | A station that has MBO features and functions enabled   |
| Multimode STA            | A station that is both Wi-Fi and cellular capable   |
| MBO multimode STA        | A station that is both Wi-Fi and cellular capable and is in addition has MBO features and functions enabled |
| Serving AP               | The AP to which an STA is associated  |
| Wi-Fi MBO Capable Device | An MBO AP or an MBO STA   |

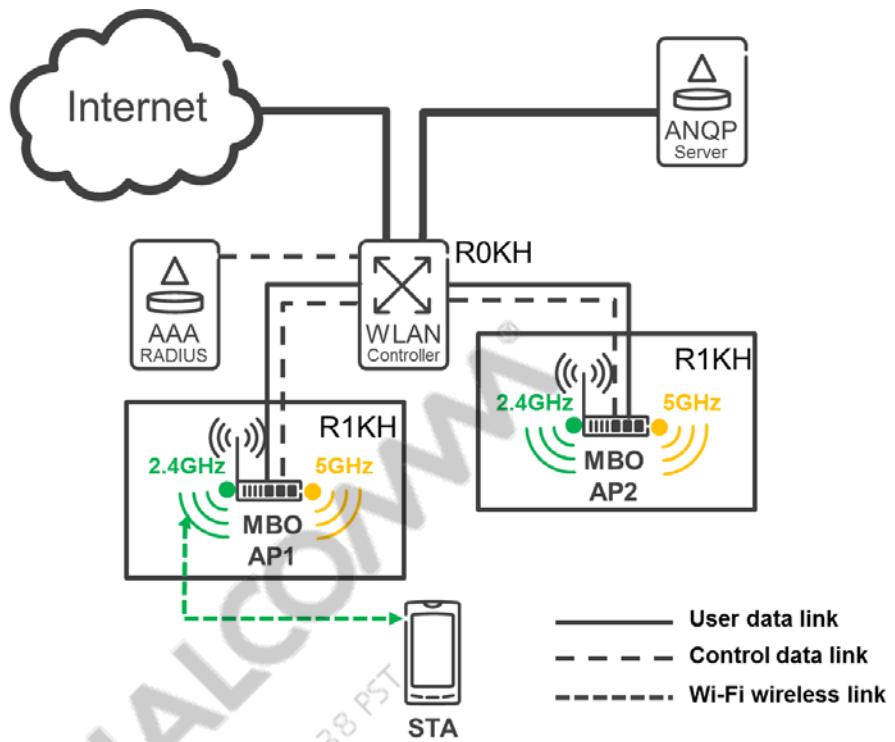
### 13.11.1 Multiband operation architecture

There are multiple components that form an MBO wireless infrastructure network, which may vary based on the wireless network deployment. This section describes some of the typical components that form an MBO wireless infrastructure network.

- Access Point (AP): An MBO wireless infrastructure network contains one (1) or more MBO APs.
- An AP that has MBO features and functions enabled is referred to as an MBO AP in this specification.
- WLAN Controller: An MBO wireless infrastructure network contains zero (0) or more WLAN Controllers that may provide centralized management and other features and functions to interconnected AP(s).
- Client Station (STA): An MBO wireless infrastructure network contains zero (0) or more client STAs. These client STAs may be single (WLAN only) or multimode (WLAN and Cellular) capable. A STA that has MBO features and functions enabled is referred to as an MBO STA in this specification.
- RADIUS Server: An MBO wireless infrastructure network contains zero (0) or more RADIUS Servers that provide Authentication, Authorization, and Accounting (AAA) services.
- ANQP Server: An MBO wireless infrastructure network contains one (1) or more ANQP servers that provide informational services about the network.
- An ANQP server in the network contains ANQP-elements or information that can be used to derive the required ANQP-elements. The information in the ANQP server can be obtained by the Access Network Query Protocol. An ANQP server can be co-located with an AP or in an external server.

### 13.11.2 Multiband operation topology

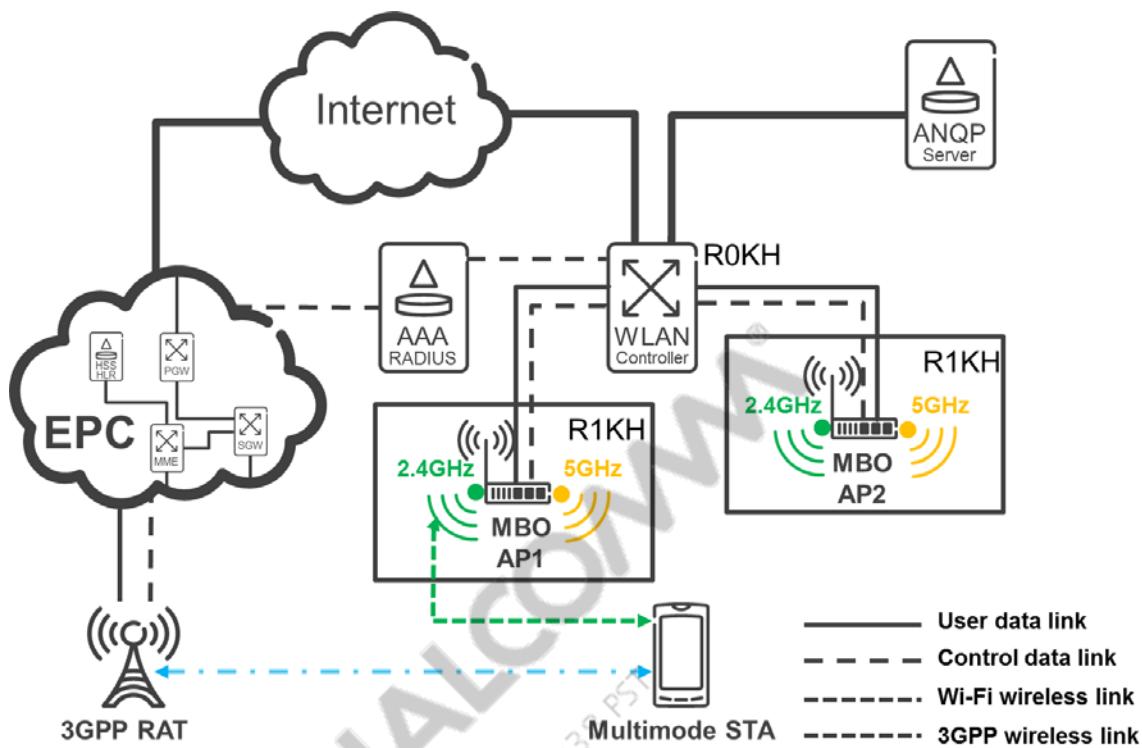
The topology of an MBO wireless infrastructure network may vary based on components and deployment requirements. This section illustrates some typical topologies.



**Figure 13-15 MBO Wireless Infrastructure Network**

In a residential wireless network, a WLAN controller and an AAA RADIUS server may not be required. The functionality provided by an ANQP server may be embedded in the WLAN AP or WLAN Controller.

In certain scenarios within the scope of this specification, an MBO AP could suggest to an MBO multimode STA that it should move its data traffic to a cellular network. The MBO Wireless Infrastructure Network with Cellular Access Network illustrates an MBO wireless infrastructure network and a cellular access network.



**Figure 13-16 MBO Wireless Infrastructure Network with Cellular Access Network**

The pairwise master key R0 key holder (R0KH) and the pairwise master key R1 key holder (R1KH) are components of the Robust Security Network Association key management. In certain scenarios, the R0KH and R1KH could be implemented in the WLAN AP. These R0KH and R1KH elements are not present if FT protocol is not supported.

### 13.11.2.1 Baseline WFA certification

This specification assumes that all the AP and STA functions and services required to pass the following WFA Certifications are implemented by Wi-Fi MBO capable Devices:

- Wi-Fi CERTIFIED™ a, OR Wi-Fi CERTIFIED™ b, OR Wi-Fi CERTIFIED™ g,
- Wi-Fi CERTIFIED WPA2™ with Protected Management Frames

### 13.11.2.2 Multiband operation-specific requirements

This section specifies the requirements for MBO APs and MBO STAs.

An MBO AP shall support the following features:

- Indicating that it supports MBO features
- Providing, on request from an MBO STA, BSS's within the ESS to which an MBO STA is recommended to associate
  - o For each BSS, include BSSID and a preference value for that BSS
  - o For each BSS, include preference value for each band and channel on that BSS

- o Include in the request an indication that the MBO STA should transfer to cellular, only if the MBO STA has indicated that it has cellular data connection available in the Cellular Data Capabilities Attribute
- Unicasting an indication of whether or not new associations can be accepted in this BSS
  - o An MBO AP may provide a time period for which an MBO STA should not attempt association to that BSS
    - § If the time period is included, the MBO AP also includes a reason code
- Broadcasting an indication if new associations are not accepted in this BSS
- Requesting that an associated MBO STA transition away from this BSS
  - o Include in the request a reason code
  - o Include in the request an indication that the MBO STA should transfer to cellular, only if the MBO STA has indicated that it has cellular data connection available in the Cellular Data Capabilities Attribute. If transfer to cellular indication is indicated, include a preference value for cellular relative to any other BSSs included in the request.
  - o Include in the request BSSs within the ESS to which an MBO STA is recommended to associate
    - § For each BSS, include BSSID and a preference value for that BSS
    - § For each BSS, include preference value for each band and channel on that BSS
- Sending a mandate to an MBO STA informing it that it is going to be disassociated
  - o Including in the mandate a time for which an MBO STA should delay any attempt to re-associate to this MBO AP
- Enabling Protected Management Frames (PMF) whenever WPA2 is enabled. An MBO AP with WPA2 enabled shall require PMF to be negotiated for use in the association when an MBO STA associates with it.

An MBO STA shall support the following features:

- Indicating whether or not it has a cellular data connection available
- Providing to an MBO AP a list of channels and bands on which it prefers not to be associated, including in each list entry:
  - o Channel and band
  - o Preference value
  - o Reason code
- Making a request to the MBO AP to provide prioritized BSSs within the ESS to which the MBO STA is recommended to associate
- Requiring PMF to be negotiated for use in the association whenever associating with an MBO AP that has WPA2 enabled.

### 13.11.2.3 Required IEEE 802.11 capabilities

This section specifies additional IEEE 802.11 capabilities that are required for support on MBO APs and MBO STAs.

An MBO AP shall support the following capabilities:

- ANQP Response, with
  - Neighbor Report ANQP-element
- Neighbor Report element, with
  - BSS Transition Candidate Preference sub-element
- BTM Request frame, with:
  - BSS Transition Candidate List Entries field
  - Disassociation Timer field
  - Preferred Candidate List Included bit
  - Disassociation Imminent bit
  - BSS Termination Included bit
- Beacon Request frame

An MBO STA or an MBO multimode STA supports the following capabilities:

- WNM-Notification Request frame
- ANQP request, with:
  - Neighbor Report ANQP-element
- Neighbor Report element
- BTM Query frame
- BTM Response frame, with:
  - BSS Transition Candidate List Entries field
  - Status Code field
  - Target BSSID field
  - BSS Termination Delay field
- Beacon Response frame

An MBO AP and an MBO STA or an MBO multimode STA may support the following capabilities:

- The FT protocol, with the over-the-air message exchange method
- ANQP response, with
  - Cellular Data Connection Preference Subtype

An MBO STA or an MBO multimode STA may support the following capabilities:

- The FT protocol, with the over-the-air message exchange method
- ANQP request, with
  - Cellular Data Connection Preference Subtype

### 13.11.3 Multiband operation protocol

The goal of the Multiband Operation (MBO) certification program is to certify features that facilitate efficient use of multiple frequency bands/channels available to Access Points (APs) and the wireless devices that may associate with them (hereafter referred to as Stations). The program is built on the fundamental premise that the AP and Stations (STAs) each have information which can aid in making the most effective selection of the spectrum band in which the STA and AP should be communicating. By exchanging this information with each other, APs and STAs enable

each other to make intelligent decisions that collectively drive the network towards more efficient use of the available spectrum.

The following sections describe the information exchanged in this regard and the mechanisms by which APs and STAs exchange the desired information. Where applicable, references to existing IEEE standards are provided. In instances where there is a need to go beyond what the IEEE standards specify, the corresponding functionality, protocols and constraints are defined.

### **13.11.3.1 Channel and Band Indication and Preference**

An MBO STA informs the AP of its channel/band capabilities by including the Supported Operating Classes element in the (Re)Association Request frame.

An MBO STA shall inform its serving MBO AP or candidate MBO AP of channels in which it will not operate, or prefers not to operate. These channels must be channels in which the MBO STA is capable of operating but prefers not to use. This information is communicated to the MBO AP when the MBO STA associates or reassociates, and when the information changes while the MBO STA is associated.

On (re)association an MBO STA shall indicate non-preferred channels by including zero (0) or more Non-preferred Channel Report Attributes in the (Re)Association Request frame. The MBO AP may store the non-preferred channels for this MBO STA.

An associated MBO STA shall indicate to the serving MBO AP that its list of non-preferred channels has changed by transmitting a WNM-Notification Request frame with Non-preferred Channel Report Subelement.

The MBO AP shall not steer the MBO STA to a channel, which the MBO STA indicated as non-operable with a value zero as preference. An MBO AP should use the information supplied in the Non-preferred Channel Report Attributes/Subelements for associated MBO STAs when selecting a new channel for the BSS and/or for requesting a BSS transition for that specific MBO STA.

Every time an MBO STA informs an MBO AP of its channel and band preferences, either through the inclusion of at least one Non-preferred Channel Report Attribute in a (Re)Association frame or the inclusion of at least one Non-preferred Channel Report Subelement in a WNM-Notification Request frame, the MBO AP shall replace all (if any) previously stored information (irrespective of Operating Class) with the most current information as indicated by the MBO STA. If the MBO AP receives an indication of channel and band preferences from an MBO STA where none of the Non-Preferred Channel Report Attributes or Subelements correspond to one of the Operating Classes supported by that STA, the MBO AP shall infer that the STA prefers to operate on all channels in that Operating Class. Similarly, if the MBO AP receives an indication of channel and band preferences from an MBO STA where none of the Non-Preferred Channel Report Attributes or Subelements correspond to particular channels in an Operating Class supported by that STA, the MBO AP shall infer that the STA prefers to operate on those particular channels.

### **13.11.3.2 Cellular Data Capability Indication**

The scope of this program requires functionality by which an MBO AP can recommend to an MBO multimode STA to transition to a cellular network. In order to avoid an MBO AP sending such an indication to a non-multimode STA, or an MBO multimode STA that does not have a

cellular data connection available, an MBO STA shall report its cellular data capability to the MBO AP using one of the following methods:

1. Probe Request frame

A non-associated MBO STA indicates its cellular data capability to an MBO AP during active scanning by transmitting a Probe Request frame to the MBO AP that contains an MBO IE with a Cellular Data Capabilities Attribute.

2. (Re)Association Request frame

A non-associated STA indicates its cellular capabilities to an MBO AP during (re) association by transmitting to said MBO AP a (Re)Association Request frame that contains an MBO IE with a Cellular Data Capabilities Subelement.

3. WNM-Notification Request frame

An associated MBO STA indicates its cellular capabilities to its serving MBO AP by transmitting an WNM-Notification Request frame that contains a Cellular Data Capabilities Attribute.

An MBO multimode STA shall update its cellular data capability to its serving MBO AP when there is a change in cellular status, e.g. one of following:

- Camping on a new cellular radio access technology
- Initial connection or a handover to a cell of a new cellular Radio Access Technology (RAT)
- Detachment or disconnection from the cellular radio technology (e.g., due to cellular coverage limitations or user action, ‘flight mode’)

An MBO AP shall use only the most recent cellular data capability indication received from the MBO STA. An MBO AP shall only send recommendation to transition to a cellular network to an MBO STA that has indicated it has a cellular data connection available.

Note that the presence of the MBO Cellular Data Capabilities Attribute is an indication of MBO capability, because all MBO STAs shall include the attribute in Probe Request frames and (Re)Association frames.

### 13.11.3.3 Beacon Report

An MBO AP shall send a Beacon Report Request frame to an associated STA whenever it requires a Beacon Report from the STA. The AP shall only send the Beacon Report Request to the STA if the STA indicates that it supports the associated RM capability.

The Beacon Report Requests can be sent with the following parameters:

- Channel Number set to
  - a specific channel (along with the appropriate operating class) or
  - 0 (along with an operating class) or
  - 255 (along with one or more AP Channel Report sub-elements)
- Measurement mode set to
  - Active or
  - Passive or
  - Beacon Table

- BSSID set to
  - o a specific BSSID or
  - o Wildcard BSSID
- Optional SSID sub-element
  - o included with the SSID of the APs of interest or
  - o not included (implying Wildcard SSID)
- Optional Reporting Detail sub-element
  - o included and set to 0 or
  - o included and set to 1 or
  - o included and set to 2
- Optional AP Channel Report sub-element
  - o one or more of these included (if Channel Number is set to 255) or
  - o not included (if Channel Number is other than 255)
- Optional Request element
  - o not included OR
  - o included and specifying a list of Element IDs

An MBO AP shall not set a measurement mode, in a Beacon Report Request to an associated STA, to a mode that the STA has not explicitly indicated support for through RM Enabled Capabilities element.

An MBO AP might use the information supplied in the Beacon Reports from associated MBO STAs as an input into an algorithm used to select a new channel for the BSS and/or for requesting a BSS transition for any associated MBO STA. The specification of such algorithms is beyond the scope of this program.

An MBO STA accepts a Beacon Report Request from its associated AP with measurement mode set to Beacon Table, and respond with a Beacon report.

An MBO STA that implements the subset of FT protocols shall implement support for active and passive measurement mode Beacon Report Requests, and indicate such support through RM Enabled Capabilities element.

An MBO STA that indicates support for active and passive Beacon Report Requests should accept a Beacon Report Request from its associated AP with measurement mode set to active or passive, and respond with a Beacon report after performing the appropriate procedures.

An MBO STA that responds to an active or passive beacon report request shall not include information about channels that do not overlap with the requested channels. A STA is allowed, but not required, to send information about overlapping channels in the Beacon Report Response that the STA sends to the AP.

**NOTE** In normal circumstances, the said MBO STA scans the requested channels and responds with the requested information. However, the STA might reject the Beacon Report request from the AP in particular circumstances where the requested scan operation impacts the quality of experience provided by the STA, such as the time at which the STA has latency-sensitive traffic, the quality of which may be impacted due to the scan operation.

### 13.11.3.4 BSS Transition Management

#### BSS Transition Information Request—STA to Candidate AP, ANQP Based

An MBO STA may request from a candidate MBO AP (that is, any AP with which it is not associated) a prioritized list of BSSs, within the AP's ESS, to which the MBO STA might transition. The ANQP request to the AP:

1. Shall include Query List ANQP-element requesting a Neighbor Report ANQP.
2. May include a Query List ANQP-element requesting a Cellular Data Connection Preference Subtype and no payload, in order to request the MBO AP to include the preference value for using cellular data connection if it is an MBO multimode STA.

On receipt of an ANQP request, the MBO AP shall respond with an ANQP response, that:

1. Shall include a Neighbor Report ANQP-element that:
  - a. Shall contain zero or more Neighbor Report elements, each of which shall contain the BSS Transition Candidate Preference sub-element, which the MBO AP considers to be candidates for MBO STA association
  - b. Shall include a Neighbor Report element for the AP's own BSS. If the AP does not wish for the STA to associate with its own BSS, it shall set the preference of its own BSS to zero.
2. May contain, when requested, the Cellular Data Connection Preference subtype that specifies the preference value of using the cellular data connection.

#### BSS Transition Information Request—STA to Serving AP, BSS Transition Management Based

An MBO STA may send a BTM Query frame to its serving MBO AP (for example, the MBO AP with which it is associated) to request a prioritized list of BSSs within the AP's ESS to which the MBO STA may transition.

On receipt of a BTM Query frame, the AP shall respond with a BTM Request frame, that:

1. Shall include a BSS Transition Candidate List Entries, that:
  - a. Contains one or more Neighbor Report elements, each of which shall contain the BSS Transition Candidate Preference sub-element
  - b. Includes a Neighbor Report element for the AP's own BSS. If the AP does not wish for the STA to associate with its own BSS, it shall set the preference of its own BSS to zero.
2. Might include an MBO IE in the BSS Transition Candidate List Entries if the MBO STA has indicated that it has cellular data connection available in the Cellular Data Capabilities Attribute. If included, the MBO IE shall contain the Cellular Data Connection Preference Attribute.

**NOTE** The AP will include cellular preference if it has information to convey about the relative preference of the cellular data connection.

## AP BSS Transition Solicitation

An MBO AP shall send an unsolicited BTM Request frame to an associated STA whenever it determines that it has to steer the STA to a new BSS (or to cellular) and before terminating the BSS.

The unsolicited BTM Request frame:

- (1) Shall indicate the recommended transition channels and BSSs using the BSS Transition Candidate List containing zero or more Neighbor Report elements. Each Neighbor Report element shall contain the BSS Transition Candidate Preference sub-element.
- (2) Shall include an MBO IE that:
  - (a) Shall contain the Transition Reason Code Attribute.
  - (b) May contain the Cellular Data Connection Preference Attribute that specifies the preference value of using the cellular data connection. The MBO AP may include in the transition request the Cellular Data Connection Request Attribute only if the MBO STA has indicated in the Cellular Data Capabilities Attribute that it has cellular data connection available.

An MBO AP shall, prior to disassociating an MBO STA, send an unsolicited BTM Request frame with Disassociation Imminent field set to one to that STA in the unsolicited BTM Request frame. When the Disassociation Imminent field is set to one, the MBO AP:

- (1) Shall set the Disassociation Timer field to the number of TBTTs that will occur prior to the MBO AP disassociating the MBO STA.
- (2) Shall instruct the MBO STA not to retry association with this BSS for the time specified in the Re-Association Delay field of the Association Retry Delay attribute, by including in the BTM Request frame an MBO IE containing an Association Retry Delay Attribute with a non-zero value in the Re-association Delay field. The count-down of the time until the STA may retry association with the same AP starts when the BTM Request is received.
- (3) Shall send a Disassociation Frame to the STA after expiry of the Disassociation Timer and no later than 2 seconds after the Disassociation Timer expires. The AP is not expected to send a Disassociation Frame if the STA has sent a Disassociation Frame to the AP beforehand.

An MBO AP shall, prior to terminating a BSS, send an unsolicited BTM Request frame to all associated MBO STAs by setting the BSS Termination Included field to one. When the BSS Termination Included field is set to one, the MBO AP:

- (1) Shall not include the Association Retry Delay Attribute in the MBO IE
- (2) Shall set the BSS Termination TSF field of the BSS Termination Duration field according to the time until the BSS is terminated
- (3) Shall terminate the BSS after the BSS Termination TSF is reached.

On receipt of an unsolicited BTM Request frame with a Disassociation Imminent bit set to one and/or BSS Termination Included field set to 1:

- (1) The MBO STA may respond with a BTM Response frame that shall contain the Status Code field indicating accept. The MBO STA may also include the Target BSSID field. When the Disassociation Imminent bit is set to one and/or the BSS Termination Included field is set to one, the STA shall not reject the Transition Management Request.
- (2) If the BTM Request frame contains a
- (3) non-zero BSS Transition Candidate List Entries field, and a BSS is available that is in the same ESS as the serving AP and is indicated in the BSS Transition Candidate List with Preference field set to a non-zero value, and the MBO STA has not received indication from the AP that BSS association or authentication would be unsuccessful, the MBO STA shall attempt to associate with another AP before the Disassociation Timer expires or the BSS Termination TSF is reached.

On receipt of an unsolicited BTM Request frame with a Disassociation Imminent bit set to zero:

- (1) If the MBO STA rejects the Transition Management Request frame (i.e. does not intend to disassociate from the MBO AP), it shall respond with a BTM Response frame that:
  - (a) Shall contain the Status Code field indicating reject.
  - (b) Shall contain the MBO IE containing the Transition Rejection Reason Code.
- (2) If the MBO STA accepts the Transition Management Request frame (i.e. intends to disassociate with the BSS) it may respond with a BTM Response frame that shall contain the Status Code field indicating accept. The MBO STA may also include the Target BSSID field.

The MBO STA may include in the BSS Transition Response frame a BSS Transition Candidate List containing Neighbor Report Elements to inform the MBO AP of BSSs it has discovered in its scanning process.

If an MBO STA has received an unsolicited BTM Request frame that contains an MBO IE with an Association Retry Delay Attribute indicating a non-zero Re-association Delay field, said MBO STA shall not attempt to associate with this BSS, after it has been disassociated, for the duration of the specified period.

### **Association Disallowed Indication**

If an MBO AP cannot accept new associations, it shall include the MBO IE with the Association Disallowed Attribute in Beacon, Probe Response and (Re)Association Response frames and shall set the value in the Status Code field in (Re)Association Response frames to a value of seventeen (17) or thirty-one (31). When an MBO STA receives a Beacon, Probe Response or (Re)Association Response frame from an MBO AP containing an MBO IE with the Association Disallowed Attribute, that MBO STA shall not send a (Re)Association Request frame or a unicast Probe Request frame to that AP until it receives a Beacon or Probe Response frame from that AP without the Association Disallowed Attribute.

The presence of the MBO IE with the Association Disallowed Attribute in any of the Beacon, Probe Response or (Re)Association Response frames shall be interpreted as an indication that the MBO AP is currently not accepting associations.

## Fast BSS Transition

In order to reduce the duration of time that connectivity is lost between an MBO STA and an ESS during a BSS transition, the MBO APs and the MBO STAs supporting WPA2-Enterprise may implement a subset of the FT protocols. Those MBO APs and MBO STAs that implement FT shall implement the Over-the-Air Protocol. The APs and STAs are not required to implement the Over-the-DS FT Protocol or FT Resource Request Protocol.

When an MBO STA intends to use Fast BSS Transition from its current MBO AP to a target MBO AP, the message exchange is performed by means of the OTA FT protocol, whereby the MBO STA communicates directly with the target MBO AP using IEEE Std. 802.11 authentication with the FT authentication algorithm. MBO APs and MBO STAs supporting only WPA2-Personal are not required to have support for Fast BSS Transition.

## MBO Capability Indication

MBO APs shall include the MBO Capability Indication Attribute in Beacon frames, Probe Response frames and (Re)Association Response frames. This attribute indicates that the AP is an MBO AP, and whether it is cellular aware or not.

### 13.11.4 Multiband Operation IE Attributes and Frame Formats

This section defines the MBO Information Element, Attributes, and frame formats for MBO APs and STAs and the MBO communication protocol.

#### Multiband Operation and Optimized Connectivity Experience Information Element

The Vendor Specific information element format shall be used to define the MBO-OCE Information Element (MBO-OCE IE) in this specification. Little endian encoding is used for multi-byte fields and subfields. The format of the MBO-OCE IE is shown in the following table:

**Table 13-10 MBO-OCE IE Format**

| Field                     | Size (Octets) | Value (Hex) | Description   |
|---------------------------|---------------|-------------|---|
| Element ID                | 1             | 0xDD        | IEEE 802.11 vendor specific information element   |
| Length                    | 1             | Variable    | Length of the following fields in the IE in octets. The Length field is a variable, and set to 4 plus the total length of the MBO Attributes. |
| OUI                       | 3             | 0x50-6F-9A  | WFA specific OUI.   |
| OUI Type                  | 1             | 0x16        | Identifying the type and version of the MBO-OCE IE  |
| MBO and/or OCE Attributes | Variable      | Variable    | One or more MBO and/or OCE Attribute(s)   |

## MBO Attributes

The MBO Attributes are defined to have a common general format consisting of a one (1) octet MBO Attribute ID field, one (1) octet Length field, and variable-length attribute-specific Information fields, as shown in the following table.

**Table 13-11 MBO Attribute General Format**

| Field                | Size (Octets) | Value (Hex) | Description                                     |  |
|----------------------|---------------|-------------|---|--|
| Attribute ID         | 1             | Variable    | Identifies the type of MBO Attribute.           |  |
| Attribute Length     | 1             | Variable    | Length of the following fields in the attribute |  |
| Attribute Body Field | Variable      | Variable    | MBO Attribute specific information fields       |  |

The following table defines the MBO Attributes and whether their support is mandatory or optional. The MBO Attributes shall be inserted in the IEEE 802.11 management frames used for MBO purposes as specified in the following table.

**Table 13-12 MBO Attributes**

| Attribute ID | Field Description                   | Management Frame                                     |  | Mandatory/Optional |     |
|--------------|-------------------------------------|--|--|--------------------|-----|
|              |                                     | AP   | STA                                    | AP                 | STA |
| 1            | MBO AP Capability Indication        | Beacon, Probe Response, and (Re)Association Response | NA                                     | M                  | NA  |
| 2            | Non-preferred Channel Report        | NA   | (Re)Association Request                | M                  | M   |
| 3            | Cellular Data Capabilities          | NA   | Probe Request, (Re)Association Request | M                  | M   |
| 4            | Association Disallowed              | Beacon, Probe Response, and (Re)Association Response | NA                                     | M                  | M   |
| 5            | Cellular Data Connection Preference | BTM Request  | NA                                     | O                  | O   |
| 6            | Transition Reason Code              | BTM Request  | NA                                     | M                  | M   |
| 7            | Transition Rejection Reason Code    | NA   | BTM Response                           | NA                 | M   |
| 8            | Association Retry Delay             | BTM Request  | NA                                     | M                  | M   |
| 9-100        | Reserved for MBO                    | NA   | NA                                     | NA                 | NA  |
| 101-150      | Reserved for OCE                    | NA   | NA                                     | NA                 | NA  |
| 0, 151-255   | Reserved                            | NA   | NA                                     | NA                 | NA  |

**NOTE** M indicates mandatory for an MBO multimode STA only, NA for other MBO STAs

### 13.11.4.1 MBO AP Capability Indication Attribute

An MBO AP Capability Indication Attribute is used by an MBO AP to indicate that it is MBO capable.

An MBO AP shall include an MBO AP Capability Indication Attribute in Beacon, Probe Response, and (Re)Association Response frames. The MBO AP Capability Indication Attribute is ordered relative to other elements in the Beacon, Probe Response, and (Re)Association frames. The following table illustrates the format of the MBO AP Capability Indication Attribute.

**Table 13-13 MBO AP Capability Indication Attribute**

| Field Name                | Size (Octets) | Value    | Description   |
|---------------------------|---------------|----------|---|
| Attribute ID              | 1             | 0x01     | Identifies the type of MBO Attribute.   |
| Attribute Length          | 1             | 0x01     | Length of the following field in the attribute in octets.                         |
| MBO Capability Indication | 1             | Variable | Each bit of this field indicates whether the corresponding capability is enabled. |

The format of the MBO AP Capability Indication field is defined in the following table:

**Table 13-14 MBO AP Capability Indication Field Values**

| Bit Position      | Description   |
|-------------------|---|
| 0x80 (MSB)        | Reserved  |
| 0x40              | When set to 1, it indicates the MBO AP is cellular aware, otherwise it indicates the MBO AP is not cellular aware |
| 0x20 – 0x01 (LSB) | Reserved  |

### 13.11.4.2 Nonpreferred Channel Report Attribute

The format of the Non-preferred Channel Report Attribute is illustrated in the following table.

An MBO STA may use the Non-preferred Channel Report Attribute to inform an AP the channels it would prefer not to or will not operate on.

One or more Non-preferred Channel Report Attributes may be included in (Re)Association Request frames. Each Non-preferred Channel Report Attribute may specify zero or more channels and their preferences.

**Table 13-15 Non-preferred Channel Report Attribute**

| Field Name       | Size (Octets) | Value            | Description   |
|------------------|---------------|------------------|---|
| Attribute ID     | 1             | 0x02             | Identifies the type of MBO Attribute.   |
| Attribute Length | 1             | 0x00 or Variable | Length of the following fields in the attribute in octets.  |
| Operating Class  | 1             | Variable         | Operating Class contains an enumerated value, specifying the operating class in which the Channel List is valid.  |
| Channel List     | Variable      | Variable         | The Channel List contains a variable number of octets, where each octet describes a single channel number.<br>An empty Channel List field indicates that the indicated Preference applies to all channels in the Operating Class.<br>Channel numbering is dependent on Operating Class. |
| Preference       | 1             | 0-255            | Indicates a preference value for the set of channels.   |
| Reason Code      | 1             | 0-255            | Indicates the reason that the MBO STA prefers not to operate in this band/channel.  |

If an MBO STA has no non-preferred channels in any Operating Class, the Attribute Length field is set to value zero (0) and Operating Class, Channel List, Preference and Reason Code fields are not present.

The values of the Preference field are defined in the following table:

**Table 13-16 Preference Field Values**

| Value | Description  |
|-------|--|
| 0     | Indicates a non-operable band/channel of the MBO STA           |
| 1     | Indicates a band/channel the MBO STA prefers not to operate in |
| 2-254 | Reserved   |
| 255   | Indicates a band/channel the MBO STA prefers to operate in     |

The value of the Reason Code field is defined in the following table.

**Table 13-17 Reason Code Field Values**

| Value | Name                    | Description   |
|-------|-------------------------|---|
| 0     | Unspecified             | Unspecified reason  |
| 1     | Co-located Interference | An unacceptable level of interference is being experienced by the MBO STA in this channel |

|       |                      |   |
|-------|----------------------|---|
| 2     | In-device Interferer | The MBO STA has another active connection in this channel, or near enough to this channel to cause operating interference |
| 3-255 | Reserved             | Reserved  |

### 13.11.4.3 Cellular Data Capabilities Attribute

The format of the MBO Cellular Data Capabilities Attribute is illustrated in the following table. The MBO Cellular Data Capabilities Attribute is included in Probe Request and (Re)Association Request frames.

**Table 13-18 MBO Cellular Capabilities Attribute**

| Field Name            | Size (Octets) | Value    | Description   |
|-----------------------|---------------|----------|---|
| Attribute ID          | 1             | 0x03     | Identifies the type of MBO Attribute.                     |
| Attribute Length      | 1             | 0x01     | Length of the following field in the attribute in octets. |
| Cellular Connectivity | 1             | Variable | Information regarding MBO STA's cellular capabilities.    |

The values of the Cellular Data Connectivity field are defined in the following table.

**Table 13-19 Cellular Data Connectivity Field**

| Value | Description                            |
|-------|--|
| 0     | Reserved                               |
| 1     | Cellular data connection available     |
| 2     | Cellular data connection not available |
| 3     | Not Cellular capable                   |
| 4-255 | Reserved                               |

**NOTE** “Cellular data connection available” indicates that the device either has an active cellular data connection or could obtain such connection (i.e. has a cellular data connection in inactive state).

### 13.11.4.4 Association Disallowed Attribute

The format of the MBO Association Disallowed Attribute is illustrated in the following table.

The MBO Association Disallowed Attribute shall be included in Beacon, Probe Response, and (Re)Association Response frames as described in the following table. The use of the MBO Association Disallowed Attribute. The MBO Association Disallowed Attribute is ordered relative to other elements in the Beacon and Probe Response frames.

**Table 13-20 Association Disallowed Attribute**

| Field Name       | Size (Octets) | Value    | Description   |
|------------------|---------------|----------|---|
| Attribute ID     | 1             | 0x04     | Identifies the type of MBO Attribute.                                   |
| Attribute Length | 1             | 0x01     | Signifies the length of the following field in the attribute in octets. |
| Reason Code      | 1             | Variable | Specifies the reason why the AP is not accepting new associations.      |

The values of the Reason Code field are defined in the following table.

**Table 13-21 Reason Code Field Values**

| Value | Description                               |
|-------|---|
| 0     | Reserved                                  |
| 1     | Unspecified reason                        |
| 2     | Maximum number of associated STAs reached |
| 3     | Air interface is overloaded               |
| 4     | Authentication server overloaded          |
| 5     | Insufficient RSSI                         |
| 6-255 | Reserved                                  |

#### 13.11.4.5 Cellular Data Connection Preference Attribute

The format of the Cellular Data Connection Preference Attribute is illustrated in following table. The MBO Cellular Data Connection Preference Attribute may be included in BTM Request frames.

**Table 13-22 Cellular Data Connection Preference Attribute**

| Field               | Size (Octets) | Value (Hex) | Description   |
|---------------------|---------------|-------------|---|
| Attribute ID        | 1             | 0x05        | Identifies the type of MBO Attribute.                       |
| Attribute Length    | 1             | Variable    | Length of the following fields of the attribute. in octets. |
| Cellular Preference | 1             | Variable    | Indicates a preference value for the cellular connection.   |

The value for the Cellular Preference field is defined in the following table.

**Table 13-23 Cellular Preference Field Values**

| Value | Description  |
|-------|--|
| 0     | Excluded. The MBO AP does not want the MBO STA to use the cellular data connection |
| 1     | The MBO AP prefers the MBO STA must not use cellular                               |
| 2-254 | Reserved   |
| 255   | The MBO AP prefers the MBO STA must use cellular                                   |

The BTM Request frame may contain the Cellular Data Connection Preference Attribute, alone or in addition to the Neighbor Report List. Use of the Cellular Data Connection Preference Attribute allows the MBO AP to indicate to the MBO STA its preference for the MBO STA to move its data traffic from WLAN to cellular. If included with a Neighbor Report List, the MBO AP can indicate the relative preference for various BSSs, channels, and the cellular data connection.

When an MBO STA, that has indicated availability of cellular data connection, receives a BSS Transition Request frame that contains an MBO IE with a Cellular Data Connection Preference Attribute, it uses the recommendation to evaluate whether the MBO STA can move its data traffic from the BSS to the cellular network and the time at which this data traffic can be moved.

An MBO STA, which did not indicate availability of cellular data connection, ignores the Cellular Data Connection Preference Attribute when this is present in the BSS Transition Request.

#### 13.11.4.6 Transition Reason Code Attribute

The format of the Transition Reason Code Attribute is illustrated in the following table. The Transition Reason Code Attribute may be included in BSS Transition Request frames.

**Table 13-24 Transition Reason Code Attribute**

| Field                  | Size (Octets) | Value (Hex) | Description   |
|------------------------|---------------|-------------|---|
| Attribute ID           | 1             | 0x06        | Identifies the type of MBO Attribute.   |
| Attribute Length       | 1             | 0x1         | Length of the following field of the attribute in octets.   |
| Transition Reason Code | 1             | Variable    | As specified in Transition Reason Codes table, identify the reason for this transition recommendation |

The MBO Reason Code Attribute is provided by an MBO AP in the BSS Transition Request frame to indicate the reason for the transition request. This attribute is mandatory in the BSS Transition Request and may be used by an MBO STA to determine its next action after receiving the BSS Transition Request from the MBO AP.

The value for the Transition Reason Code field is defined in the following table.

**Table 13-25 Transition Reason Code Field Values**

| Transition Reason Code | Description  |
|------------------------|--|
| 0                      | Unspecified  |
| 1                      | Excessive frame loss rate  |
| 2                      | Excessive delay for current traffic stream   |
| 3                      | Insufficient bandwidth for current traffic stream  |
| 4                      | Load balancing   |
| 5                      | Low RSSI   |
| 6                      | Received excessive number of retransmissions   |
| 7                      | High interference  |
| 8                      | Gray zone<br>Imbalance between the PHY operating margin in the downlink direction compared to the uplink direction can result in a “gray” zone of coverage in which MBO STAs can become “stalled” in certain states such as: <ul style="list-style-type: none"><li>■ Mobile is 802.11 authenticated, but not associated</li><li>■ Mobile is 802.11 associated, but not EAP authenticated</li><li>■ Mobile is unable to obtain an IP address through DHCP</li><li>■ Mobile is unable to perform name resolution through DNS</li></ul> If the serving MBO AP can detect that an MBO STA is in a gray zone, it must attempt to enable the device to transition. |
| 9                      | Transitioning to a Premium AP  |
| 10-255                 | Reserved   |

#### 13.11.4.7 Transition Rejection Reason Code Attribute

The format of the Transition Rejection Reason Code Attribute is illustrated in the following table. The attribute may be included in BTM Response frames.

**Table 13-26 Transition Rejection Reason Code Attribute**

| Field                            | Size (Octets) | Value (Hex) | Description   |
|----------------------------------|---------------|-------------|---|
| Attribute ID                     | 1             | 0x07        | Identifies the type of MBO Attribute.   |
| Attribute Length                 | 1             | 0x1         | Length of the following field of the attribute in octets.   |
| Transition Rejection Reason Code | 1             | Variable    | As specified in Transition Rejection Reason Codes table, identify the reason that the MBO STA rejected this transition recommendation |

The values of the Transition Rejection Reason Code field are defined in the following table:

**Table 13-27 Transition Rejection Reason Code Field Values**

| Transition Rejection Reason Code | Description  |
|----------------------------------|--|
| 0                                | Unspecified  |
| 1                                | Excessive frame loss rate expected by the MBO STA if it transitions to the suggested candidate BSS(s) in the BTM Request frame   |
| 2                                | Excessive delay for current traffic stream would be incurred by BSS transition at this time  |
| 3                                | Insufficient QoS capacity for current traffic stream expected by the MBO STA if it transitions to the suggested candidate BSS(s) in the BTM Request frame  |
| 4                                | Low RSSI in frames being received by the MBO STA from to the suggested candidate BSS(s) in the BTM Request frame   |
| 5                                | High interference expected by the MBO STA if it transitions to the suggested candidate BSS(s) in the BTM Request frame   |
| 6                                | Service Availability – the MBO STA expects that services it needs which are available at its serving AP will not be available if it transitions to the suggested candidate BSS(s) in the BTM Request frame |
| 7-255                            | Reserved   |

#### 13.11.4.8 Association Retry Delay Attribute

The format of the Association Retry Delay Attribute is illustrated in the following table. The attribute may be included in BTM Request frames.

**Table 13-28 Association Retry Delay Attribute**

| Field Name           | Size (Octets) | Value    | Description   |
|----------------------|---------------|----------|---|
| Attribute ID         | 1             | 0x08     | Identifies the type of MBO Attribute.   |
| Attribute Length     | 1             | 0x02     | Length of the following field in the attribute in octets.   |
| Re-association Delay | 2             | Variable | A mandatory field that indicates the number of seconds which the MBO STA must wait before trying to (re-)associate with the BSS.<br>This timer starts when the BTM Request containing this field is received. |

#### 13.11.4.9 MBO ANQP-elements

The MBO ANQP-elements follows the definition specified in Vendor Specific ANQP-element. The following are the elements:

**Table 13-29 MBO ANQP-elements**

| Name     | Size (Octets) | Value      | Description  |
|----------|---------------|------------|--|
| InfoID   | 2             | 56797      | ANQP-element definitions (value 56797).  |
| Length   | 2             | Variable   | Length of the OUI field plus the length of the Vendor Specific content   |
| OUI      | 3             | 0x50-6F-9A | The OUI is a 3-octet field. The OUI field is set to the value used by the WFA, (value 0x50-6F-9A)                            |
| OUI Type | 1             | 0x12       | MBO ANQP vendor specific element ID  |
| Subtype  | 1             | Variable   | The Subtype field is a 1-octet field whose value identifies the MBO ANQP-element.  |
| Payload  | Variable      | Variable   | Vendor Specific Content field is a variable length field whose content is defined by the entity identified in the OUI field. |

The following table defines the Subtype values:

**Table 13-30 MBO ANQP-element Subtype Values**

| Element Name                        | Subtype Value | Description (section)  |
|-------------------------------------|---------------|--|
| Reserved                            | 0             | N/A  |
| MBO Query List                      | 1             | Provides a list of identifiers of MBO ANQP-elements for which the requesting STA is querying in an MBO ANQP query.   |
| Cellular Data Connection Preference | 2             | Used by an MBO AP to provide an MBO STA with a preference value to move to cellular, equivalent to a BSS Transition Candidate Preference in the Neighbor Report. |

#### **13.11.4.10 MBO Query List**

The MBO Query list provides a list of identifiers of MBO ANQP-elements for which the requesting STA is querying in an MBO ANQP query. The MBO Query List ANQP-element is included in a GAS Query Request frame. The MBO Query List ANQP-element shall be used in a GAS Query Request to request MBO ANQP-elements. Both the ANQP Query List and the MBO Query List may be included in a single GAS Query Request

The format of the MBO Query List payload is provided the following figure:

|                     |                                   |                                   |
|---------------------|-----------------------------------|-----------------------------------|
| MBO ANQP element #1 | MBO ANQP element #2<br>(optional) | MBO ANQP element #3<br>(optional) |
|---------------------|-----------------------------------|-----------------------------------|

Octets: 1 1 1

The value of each MBO ANQP element Subtype field is a Subtype. A Subtype is included in the MBO Query List element to indicate that the STA device performing the GAS Query Request is requesting that the MBO ANQP element corresponding to that Subtype be returned in the GAS Query Response. The Subtypes included in the MBO Query List element shall be ordered by monotonically increasing Subtype value.

#### 13.11.4.11 Cellular Data Connection Preference Subtype

The Cellular Data Connection Preference Subtype is a MBO ANQP-element that is used by an MBO AP to provide an MBO STA with a preference value to move to cellular, equivalent to a BSS Transition Candidate Preference in the Neighbor Report.

The Cellular Data Connection Preference Subtype is a one octet payload with values.

#### 13.11.4.12 Neighbor Report in ANQP

The Neighbor Report ANQP-element provides only one neighbor report about one neighboring AP. If the MBO AP wants to send multiple neighbor reports to the STA, it can include multiple neighbor report ANQP-elements into the GAS Response frame.

#### 13.11.4.13 WNM-Notification

The format of WNM-Notification is illustrated in the following figure:

| Category | Action | Dialog Token | Type | Optional Subelements |
|----------|--------|--------------|------|----------------------|
|----------|--------|--------------|------|----------------------|

**Figure 13-18 WNM-Notification Request frame format**

The Type field is set to value 0xDD (221 – Vendor Specific). The Category, Action, Dialog Token are set to conforming values. The Optional Subelements field contains Non-preferred Channel Report subelement or Cellular Capabilities subelement.

##### Non-preferred Channel Report Subelement

The format of the Non-preferred Channel Report subelement is defined in Figure 4. One or more Non-preferred Channel Report Subelements may be included in Optional Subelements field in WNM-Notification Request frame. Each Non-preferred Channel Report Subelement may specify zero or more channels and their preferences as defined in the following table.

**Table 13-31 Non-preferred Channel Report Subelement**

| Field Name    | Size (Octets) | Value            | Description   |
|---------------|---------------|------------------|---|
| Subelement ID | 1             | 0xDD             | The Subelement ID is set to value 0xDD (221 – Vendor Specific). |
| Length        | 1             | 0x04 or Variable | Length of the following fields in the subelement in octets.     |

|                 |          |          |   |
|-----------------|----------|----------|---|
| OUI             | 3        | 0x506F9A | The OUI is set to value 0x506F9A, as used by the Wi-Fi Alliance.  |
| OUI Type        | 1        | 0x02     | The OUI Type is set to value 0x02 as used by the Wi-Fi Alliance for Non-preferred Channel Report.   |
| Operating Class | 1        | Variable | Operating Class contains an enumerated value, specifying the operating class in which the Channel List is valid.  |
| Channel List    | Variable | Variable | The Channel List contains a variable number of octets, where each octet describes a single channel number.<br>An empty Channel List field indicates that the indicated Preference applies to all channels in the Operating Class.<br>Channel numbering is dependent on Operating Class. |
| Preference      | 1        | 0-255    | Indicates a preference value for the set of channels.   |
| Reason Code     | 1        | 0-255    | Indicates the reason that the MBO STA prefers not to operate in this band/channel.  |

If an MBO STA has no non-preferred channels in any Operating Class, the Length field is set to value four (4) and Operating Class, Channel List, Preference and Reason Code are not present.

Note that Non-preferred Channel Report subelement and Non-preferred Channel Report Attribute are two different structures. Non-preferred Channel Report subelement is used in the WNM-Notification Request frame and Non-preferred Channel Report attribute is used in (Re)Association frames.

### Cellular Capabilities Subelement

The format of the Cellular Capabilities b is defined in the following table. The subelement is used in Optional Subelements field in WNM-Notification Request frame.

**Table 13-32 Table 4-23: Cellular Data Capabilities Subelement**

| Field Name    | Size (Octets) | Value            | Description   |
|---------------|---------------|------------------|---|
| Subelement ID | 1             | 0xDD             | The Subelement ID is set to value 0xDD (221 – Vendor Specific). |
| Length        | 1             | 0x00 or Variable | Length of the following fields in the subelement in octets.     |
| OI            | 3             | 0x506F9A         | The OI is set to value 0x506F9A, as used by the Wi-Fi Alliance. |

|                       |   |          |   |
|-----------------------|---|----------|---|
| OUI Type              | 1 | 0x03     | The OUI Type is set to value 0x03 as used by the Wi-Fi Alliance for Cellular Data Capabilities. |
| Cellular Connectivity | 1 | Variable | The Cellular Data Connectivity provides information regarding MBO STA's cellular capabilities.  |

Note that Cellular Data Capabilities sub-element and Cellular Capabilities attribute have two different structures. The Cellular Data Capabilities sub-element is used in the WNM-Notification Request frame and the Cellular Data Capabilities Attribute is used in all other frames.

## 13.12 Host Tx flow control

The host Tx flow control enables better throughput in the Tx data path. Without the host Tx flow control, when there is heavy traffic load, the firmware gets the packets from the host and drops them.

With the host Tx flow control and the firmware back pressure enabled, the Tx data path on the host starts queuing the packets as and when it runs out of OL Tx descriptors. The following two APIs queue the packets:

- `ol_tx_check_enqueue_flow_ctrl`
- `ol_tx_enqueue_flow_ctrl`

The packets are later de-queued in the Tx completion handler. The `ol_tx_ll_sched` API de-queues the packets.

The “`iwpriv ath0 txrx_fs_stats 8`” command displays the number of:

- Tx packets sent directly without queuing
- Tx packets queued
- Packets dropped due to queue overflow

## 13.13 Descriptor configuration for FW

The host driver configures the number of descriptors the firmware can have at module load time, using the “`max_descs`” module parameter. The parameter is a 16-bit integer with the value in the range of 1424 to 2160.

The module parameter can be set using the following UCI commands.

```
uci set wireless.qcawifi=qcawifi
uci set wireless.qcawifi.max_descs= <1424 to 2160>
uci commit
wifi
```

## 13.14 Dynamic beacon

**NOTE** The dynamic beacon functionality is supported only in QCA\_Networking\_2016.SPF.4.0 release; it is replaced by the enhanced dynamic beacon functionality, starting with QCA\_Networking\_2016.SPF.5.0 release.

| Feature        | IPQ4019.ILQ.4.0 |             |             |             | IPQ8064.ILQ.4.0 |             |            |             |             | QCA9531.ILQ.4.0 |             |             |             | QCA9558.ILQ.4.0 |             |             | QCA9563.ILQ.4.0 |             |             |             |   |   |
|----------------|-----------------|-------------|-------------|-------------|-----------------|-------------|------------|-------------|-------------|-----------------|-------------|-------------|-------------|-----------------|-------------|-------------|-----------------|-------------|-------------|-------------|---|---|
|                | IPQ<br>4019     | QCA<br>9886 | QCA<br>9984 | QCA<br>9889 | QCA<br>9984     | QCA<br>9980 | AR<br>9380 | QCA<br>9880 | QCA<br>9889 | QCA<br>9880     | QCA<br>9889 | QCA<br>9531 | QCA<br>9886 | QCA<br>9984     | QCA<br>9558 | QCA<br>9880 | QCA<br>9889     | QCA<br>9563 | QCA<br>9880 | QCA<br>9886 |   |   |
|                | 3.14            |             |             |             | 3.14            |             |            |             |             | 3.3.8           |             |             |             |                 |             |             |                 |             |             |             |   |   |
| Dynamic Beacon | ✓               | ✓           | ✓           | ✓           | ✓               | ✓           | ✓          | ✓           | ✓           | ✓               | ✓           | ✓           | ✓           | ✓               | ✓           | ✓           | ✓               | ✓           | ✓           | ✓           | ✓ | ✓ |

This feature is used to create a dedicated VAP which is active but does not send beacon always and enable/disable beacon on certain condition. Create a dedicated VAP with hidden SSID for station devices easy on-boarding and set up. For example: customer's Repeater and IoT devices.

The specific hidden SSID in the Router is pre-configured in customer's specific station devices and do probe request to connect to the Router hidden SSID, Router will then push all the configuration profiles to the repeater.

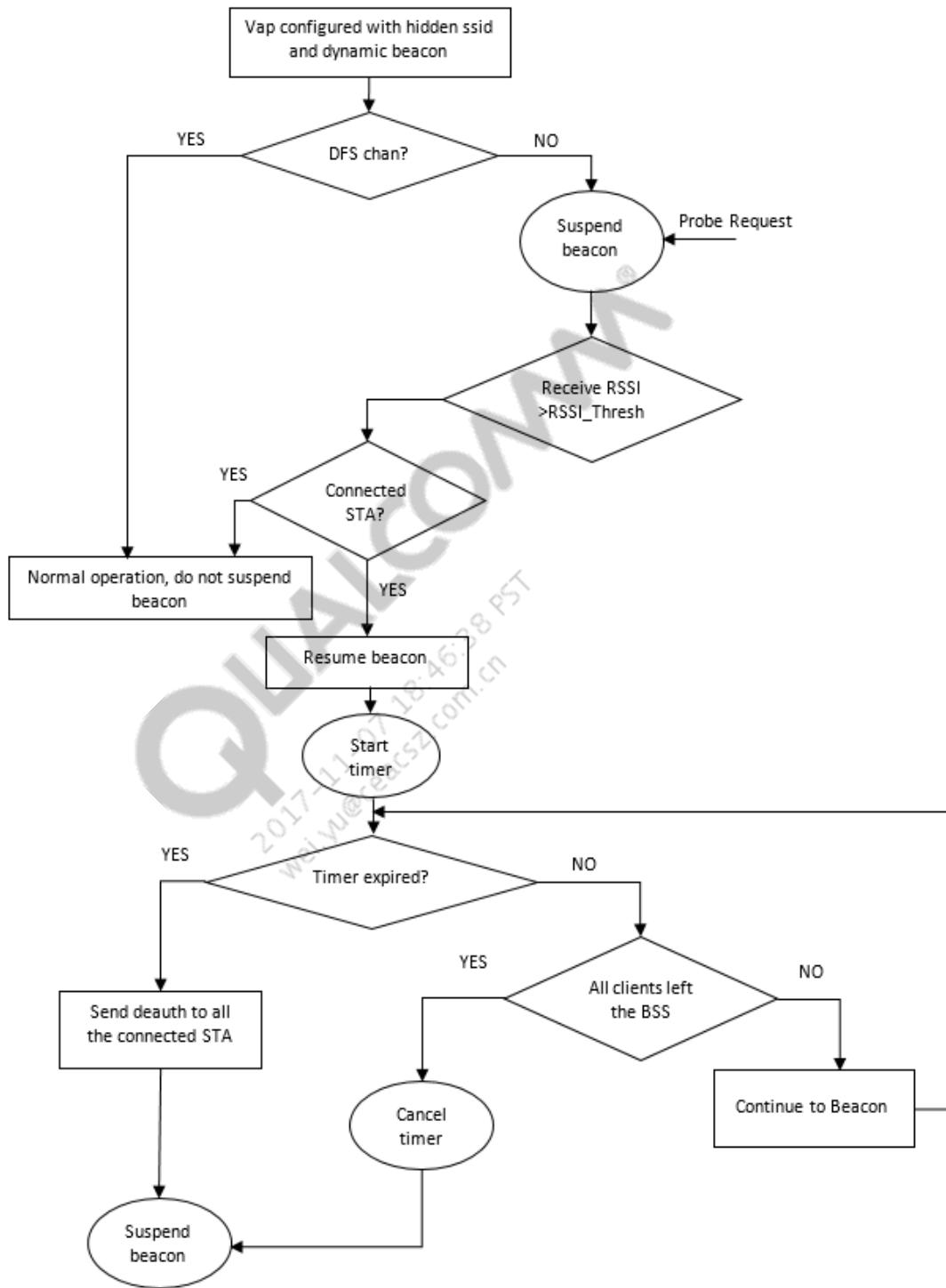
1. VAP is the same as other AP VAP, except it does not send beacon.
2. VAP only sends beacon (hidden SSID beacon) with all the following conditions met:
  - a. STA sends unicast probe request with correct SSID
  - b. STA probe request RSSI is high enough (RSSI threshold should be configurable)
3. VAP stops sending beacon with either of the following conditions met
  - a. All STAs disconnect with AP
  - b. After a timeout value (timeout should be configurable), AP should also kick off STA after timeout
  - c. When multiple STAs connected, timeout value should be counted since the last STA connected.

### Implementation

1. During VAP init check if the channel is DFS then set the runtime dynamic feature support flag to 0, so that the AP VAP behaves as normal AP VAP. For non DFS channel the run time dynamic beacon flag is set to 1 and suspend the beacon.

2. On receiving probe request, check if the feature is enabled and runtime dynamic beacon flag is set on current VAP, if it is not set, follow the normal 802.11 AP VAP flow. If the flag is set, and if the RSSI is greater than RSSI threshold, then resume beaconing and start a timer with the timeout value. Within the timeout AP can accept the connection from STA.
3. Within the timeout when all the clients leave (assoc number is 0) suspend beacon and cancel the timer.
4. When the timer is expired, kick off all STA under this VAP, then suspend the beacon.

QUALCOMM®  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

**Figure 13-19 Flow of Suspend and Resume beacon**

### Commands

Enable/disable the dynamic beacon feature.

```
iwpriv athX dynamicbeacon 1/0
uci set wireless.@wifi-iface[X].dynamicbeacon=0/1
```

Set/Get the rss threshold, default 60.

```
iwpriv athX db_rssi_thr 55 / iwpriv athX g_db_rssi_thr
uci set wireless.@wifi-iface[X].db_rssi_thr =55
```

Set/Get the timeout of timer, default 30.

```
iwpriv athX db_timeout 30 / iwpriv athX g_db_timeout
uci set wireless.@wifi-iface[X].db_timeout=30
```

## 13.15 Enhanced dynamic beacons

**NOTE** Until QCA\_Networking\_2016.SPF.4.0 release, the dynamic beacon functionality caused a dedicated VAP on which this capability was enabled to send beacon frames to STAs after the probe requests were received with the RSSI of the probe request being higher than the configured RSSI threshold value. Beacons were suspended after a timeout for disconnecting the STA from AP was exceeded or when all STAs left the network. Starting with QCA\_Networking\_2016.SPF.5.0 release, the dynamic beacon functionality is enhanced to cause the dedicated VAP to send beacon frames only after the STA association with the AP is completed; beacons are not sent immediately after receiving STA probe requests. The behavior of suspension of beacons remains unchanged.

This enhanced dynamic beacon feature creates a dedicated VAP that is active, but does not always send beacons; instead, the transmission of beacons is enabled and disabled, based on certain conditions. A dedicated VAP with hidden SSID is created for easy on-boarding and set up of station devices (for example, repeaters of customers and IoT devices). The specific hidden SSID in the router is preconfigured in customer-specific station devices and performs probe requests to connect to the router hidden SSID. Thereafter, the router pushes all the configuration profiles to the repeater.

### Guidelines for working of enhanced dynamic beacons

The following guidelines apply to the working of the enhanced dynamic beacon functionality.

1. VAP is the same as other AP VAP, except it does not send beacon.
2. VAP sends probe responses only when all of the following conditions are satisfied:
  - a. STA sends unicast probe request with correct SSID.
  - b. STA probe request RSSI is high enough (RSSI threshold must be configurable).
3. VAP sends beacons (hidden SSID beacon) only when the following conditions are satisfied:
  - a. At least one station has associated with AP successfully.
  - b. AP sends associate response frame with reason association-complete to station.
4. VAP stops sending beacons when any one of the following conditions is satisfied:
  - a. All STAs disconnect from AP.

- b. After a timeout value (timeout must be configurable), AP also disconnects STA.
- c. When multiple STAs are connected, timeout value is computed from the time at which the last STA was connected.

## Workflow of enhanced dynamic beacons

The following sequence of events occur with the enhanced dynamic beacon functionality:

1. During VAP initialization, a check is performed to determine whether the channel is a dynamic frequency selection (DFS-capable) channel. If so, the runtime dynamic feature support flag is set to 0 so that the AP VAP behaves as a normal AP VAP. For non-DFS channels, the runtime dynamic beacon flag is set to 1 and the beacon is suspended.
2. After receiving a probe request, a check is performed to determine whether the dynamic beacon feature is enabled (by entering the `iwpriv athX dynamicbeacon 1` command) and runtime dynamic beacon flag is set on current VAP.
  - a. If the dynamic beacon flag is not set, the normal 802.11 AP VAP flow occurs.
  - b. If the dynamic beacon flag is set, and if the RSSI is greater than RSSI threshold, AP sends the probe response, but does not resume the beacon.
3. During association, when AP sends association response frames with reason as association-complete to station, VAP resumes the sending of beacons and starts the timer.
4. Within the timeout period, when all the clients leave (association number is 0), VAP suspends the beacon and cancels the timer.
5. When the timer is expired, all STAs under this VAP are disconnected, and then VAP suspends the beacon.

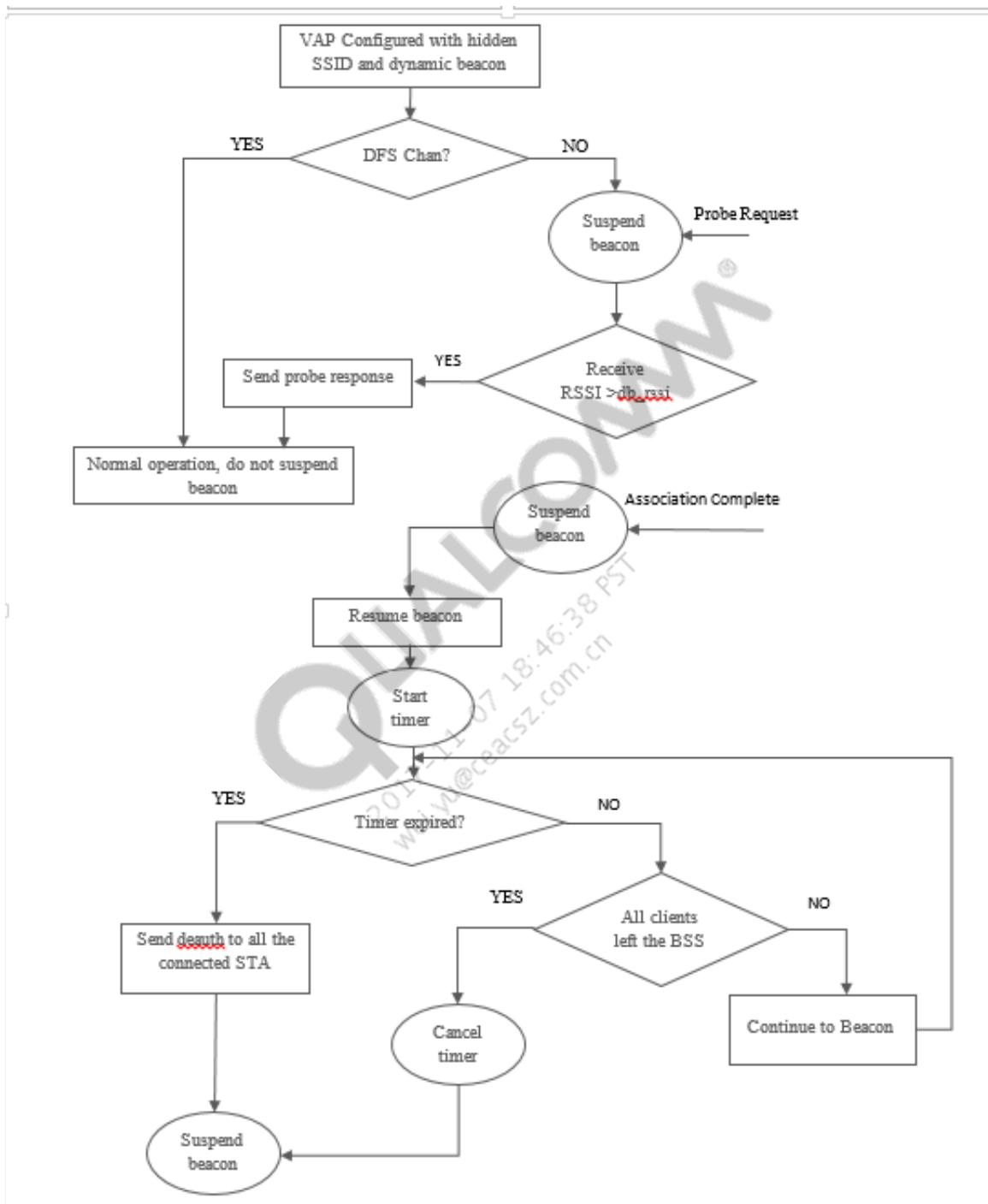


Figure 13-20 Workflow of suspension and resumption of beacons

## 13.16 Beacon rate control per SSID for offload mode

A functionality to support rate-control for beacons per SSID for offload chipsets is introduced. The valid basic rates to be set for beacon frames (in kbps) are as follows:

- 2 GHz radio: 1000, 2000, 5500, 6000, 11000, 12000, 24000
- 5 GHz radio: 6000, 12000, 24000

Beacon rate value in wal\_vdev that is set is used to send out beacons. This variable is initialized to invalid rate code, which indicates that the default rate in respective bands is used. Host sends the beacon rate code to be used to firmware. All processing-related to the user iwpriv input and processing of the correct valid rate code to use is performed by the host. Firmware accepts Complementary Code Keying (CCK) rates for 2 GHz and 801.11a rates for 2 GHz 802.11g and 5 GHz as valid rates for beacon. If invalid rate value is received from host, firmware sets the lowest rate for the corresponding band as the default rate.

User is allowed to enter any value for beacon rate in the range of 1000 Kbps to 24000 Kbps for 2 GHz VAP and in the range of 6000 Kbps to 24000 Kbps for 5 GHz VAP. If the rate is higher than these defined ranges, the beacon rate is set to the previously specified valid rate. If no rate was previously specified, the beacon rate is set to the default rate.

Assume that a user is passing 5400 (in case 2 GHz vap). Because this rate value is an allowed rate but not a valid basic rate, the driver attempts to search for the next higher valid basic rate (that is, 5500 and set 5500 Kbps as beacon rate).

Assume that a user disables all the higher rates from 5500 to 24000 (that can be done by using iwpriv athX dis\_legacy bitmask) and user sets 5400 as beacon rate. Because it does not find any higher rates available in the rate table, the driver selects the lowest available rate as beacon rate.

Only beacon rate is modified, whereas all the other management and control frames are sent at the default rate. AP supports all the rates (even the lower rates) according to the mode in which it is configured, regardless of the beacon rate change.

To set any rate as beacon rate, use the following command:

```
iwpriv athX set_bcn_rate <rate in Kbps>
```

Example: iwpriv ath1 set\_bcn\_rate 6000 will set the beacon rate to 6 Mbps.

To get the beacon rate, use the following command:

```
iwpriv athX get_bcn_rate
```

By default for 2 GHz radio: iwpriv ath1 get\_bcn\_rate will give 1000

By default for 5 GHz radio: iwpriv ath1 get\_bcn\_rate will give 6000

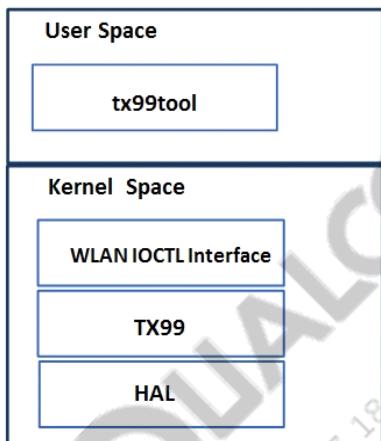
## 13.17 TX99 Mode

TX99 is continuous transmit mode. In other words, in this mode Wi-Fi radio continuously transmitting a given data packet. This mode of transmit is used to tune radio by hardware/system team. This feature in driver makes these tool accessible as part of regular driver image. Prior to this feature, a special driver and related packages need to be loaded on AP to run TX99. However, after

this feature, TX99 can be triggered from command line utility “tx99tool/athtestcmd” tool without loading any special package. Note that there are separate tools for 11N radio called “tx99tool” and for 11AC radio tool is called “athtestcmd”. The reason for separate tool is because architecture of these two interfaces are fundamentally different.

### 13.17.1 TX99 for 11n Radio

The radio needs to be put in test mode in order to transmit continuously. “tx99tool” utility is used to send the command to the driver to configure radio in test mode (described in picture below).



**Figure 13-21 TX99 11n Mode Overview**

The Qualcomm Technologies LMAC provides an LMAC API to enable/disable the TX99 feature of the WLAN driver. To enable this feature at configuration time, ATH\_SUPPORT\_TX99 should be set to 1. The format of this API is shown below:

```
sc_ops->tx99_ops(sc, int cmd, struct tx99_wcmd *tx99_wcmd);
```

Related data structures are defined as follows:

```

typedef enum {
    TX99_WCMD_ENABLE, /* start */
    TX99_WCMD_DISABLE, /* stop */
    TX99_WCMD_SET_FREQ,
    TX99_WCMD_SET_RATE,
    TX99_WCMD_SET_POWER,
    TX99_WCMD_SET_TXMODE,
    TX99_WCMD_SET_CHANMASK,
    TX99_WCMD_SET_TYPE,
    TX99_WCMD_SET_TESTMODE,
    TX99_WCMD_GET
} tx99_wcmd_type_t;

typedef struct tx99_wcmd_data {
    u_int32_t freq; /* tx frequency (MHz) */
    u_int32_t htmode; /* tx bandwidth (HT20/HT40) */
    u_int32_t htext; /* extension channel offset */
    u_int32_t rate; /* Kbits/s */
    u_int32_t power; /* (dBm) */
    u_int32_t txmode; /* wireless mode */
}
  
```

```

u_int32_t chanmask; /* tx chain mask */
u_int32_t type;
u_int32_t testmode; /* tx data pattern / recv only */
} tx99_wcmd_data_t;
typedef struct tx99_wcmd {
char if_name[IFNAMSIZ];/**< Interface name */
tx99_wcmd_type_t type; /**< Type of wcmd */
tx99_wcmd_data_t data; /**< Data */
} tx99_wcmd_t;

```

When this TX99 API is called, both type and data fields should be supplied in tx99\_wcmd with proper values except for TX99\_WCMD\_ENABLE and TX99\_WCMD\_DISABLE operations.

## 13.17.2 TX99 ioctl Option

### 13.17.2.1 TX99\_WCMD\_SET\_FREQ

The following fields need to be set when setting the channel frequency:

freq. IEEE channel number

htmode. 0: CWM\_MODE20; 2: CWM\_MODE40;

htext. 0: Legacy; 1: extension is plus; 2: extension is minus

### 13.17.2.2 TX99\_WCMD\_SET\_RATE

The following fields need to be set when setting the rate:

rate. PHY transmission rate in Kbps

### 13.17.2.3 TX99\_WCMD\_SET\_TXMODE

The following fields need to be set when setting the wireless mode:

txmode. 0: 11A; 1: 11B; 2: 11G; 5: 11NAHT20; 6: 11NGHT20; 7: 11NAHT40+;  
8: 11NAHT40-; 9: 11NGHT40+; 10: 11NGHT40-;

### 13.17.2.4 TX99\_WCMD\_SET\_CHANMASK

The following fields need to be set when setting the channel mask:

chanmask. 01 – 11 for 2x2; 001 – 111 for 3x3;

### 13.17.2.5 TX99\_WCMD\_SET\_TESTMODE

The following fields need to be set when setting the test mode:

testmode. 0: PN9 data; 1: All zeros; 2: All ones; 3: RX only; 4: Single carrier mode

### 13.17.2.6 TX99\_WCMD\_SET\_POWER

The following fields need to be set when setting the tx power:

txpower.: Transmit power in dB.

QUALCOMM®  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

### 13.17.3 TX99 Command Guide

#### 13.17.3.1 Setting the frequency parameters

Use the following command to set the frequency parameters:

```
tx99tool wifi0 set freq [freq_ieee] bandwidth [bandwidth] htext [htext]
```

freq\_ieee: channel No.

bandwidth: 0: 20 MHz; 2: 40 MHz

htext: 0: legacy; 1: when the extension is plus; 2: when the extension is minus

Example

```
tx99tool wifi0 set freq 36 bandwidth 0 htext 0
```

#### 13.17.3.2 Setting the rate

Use the following command to set the rate:

```
tx99tool wifi0 set rate [kbps]
```

Example

To set the rate to 6 mbps

```
tx99tool wifi0 set rate 6000
```

#### 13.17.3.3 Setting the PHY mode

Use the following command to set the PHY mode:

```
tx99tool wifi0 set txmode [mode]
```

The following are the PHY modes:

- 0:11a
- 1:11b
- 2:11g
- 5:11naht20
- 6:11g20
- 7:11na40+
- 8:11na40-
- 9:11ng40+
- 10:11ng40-

Example

To set the phy mode to 11a:

```
tx99tool wifi0 set txmode 0
```

### 13.17.3.4 Setting the Tx chainmask

Use the following command to set the Tx chainmask:

```
tx99tool wifi0 set txchain [txchainmask]
```

The following are the TX chain mask parameters:

```
txchainmask 1: 1x1: 3: 2X2; 7: 3X3
```

Example

```
tx99tool wifi0 set txchain 3
```

### 13.17.3.5 Setting the test mode

Setting the test mode parameters based on the following:

```
tx99tool wifi0 set testmode [test mode]
```

The following are the test modes:

- 0: Transmit PN9 data
- 1: Transmit All zeros
- 2: Transmit All ones
- 3: Receive only
- 4: Single carrier mode

Example

```
tx99tool wifi0 set testmode 0
```

### 13.17.3.6 Setting the Tx Power

Setting the Tx power using the following command:

```
tx99tool wifi0 set pwr [tx power] TX power is in dB.
```

Example

```
tx99tool wifi0 set pwr 20
```

### 13.17.3.7 Enabling Tx99

Use the following command to enable TX99:

```
tx99tool wifi0 start
```

### 13.17.3.8 Stopping Tx99

Use the following command to stop the TX99:

```
tx99tool wifi0 stop
```

### 13.17.4 TX99 for 802.11AC Radio

The radio needs to be put in test mode in order to transmit continuously. Driver needs to load special firmware image called “utf.bin” to put radio into test mode. Once AP is configured to run TX99, regular AP functionalites are disabled. To start TX99 on a particular channel, frequency, and so on, the athtestcmd utility is used to send IOCTL to driver. Command details are provided in [Section 13.17.2.1](#)

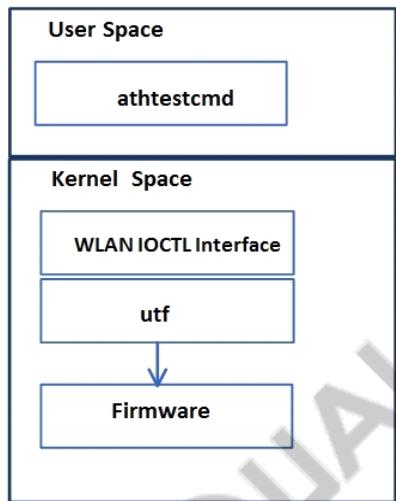


Figure 13-22 TX99 11ac Mode Overview

### 13.17.4.1 attestcmd command guidelines

**Table 13-33 attestcmd Frequently Used Options**

| Command option | Example   | Description  | Parameter                                     |                 |
|----------------|---|--|---|-----------------|
| --txfreq       | --txfreq <Tx channel or freq>   | Frequency or Channel to output the waveform (in MHz) | 5180,5805,and so on...<br>40,161,and so on... |                 |
| --txrate       | --txrate <rate index><br>(<0-3> = 1-11Mbps,<br><4-11> = 6 -54 Mbps,<br><12-19> = 6.5 -65Mbps,<br>and so on) | Date rate to output the waveform                     | Rate index                                    | Phy Rate (Mbps) |
|                |   |  | 1   | 1               |
|                |   |  | 2   | 5.5             |
|                |   |  | 3   | 11              |
|                |   |  | 4   | 6               |
|                |   |  | 5   | 9               |
|                |   |  | 6   | 12              |
|                |   |  | 7   | 18              |
|                |   |  | 8   | 24              |
|                |   |  | 9   | 36              |
|                |   |  | 10  | 48              |
|                |   |  | 11  | 54              |
|                |   |  | 12  | 6.5             |
|                |   |  | 13  | 13              |
|                |   |  | 14  | 19.5            |
|                |   |  | 15  | 26              |
|                |   |  | 16  | 39              |
|                |   |  | 17  | 52              |
|                |   |  | 18  | 58.5            |
|                |   |  | 19  | 65              |
|                |   |  | 20  | 13              |
|                |   |  | 21  | 26              |
|                |   |  | 22  | 39              |

**Table 13-33 athtestcmd Frequently Used Options (cont.)**

| <b>Command option</b> | <b>Example</b>  | <b>Description</b>                    | <b>Parameter</b> |              |
|-----------------------|---|---------------------------------------|------------------|--------------|
| txrate                | --txrate <rate index><br>(<0-3> = 1-11Mbps,<br><4-11> = 6 -54 Mbps,<br><12-19> = 6.5 -65Mbps,<br>and so on) | Date rate to output the waveform      | 23               | 52           |
|                       |   |                                       | 24               | 78           |
|                       |   |                                       | 25               | 104          |
|                       |   |                                       | 26               | 117          |
|                       |   |                                       | 27               | 130          |
|                       |   |                                       | 28               | 13.5         |
|                       |   |                                       | 29               | 27.0         |
|                       |   |                                       | 30               | 40.5         |
|                       |   |                                       | 31               | 54           |
|                       |   |                                       | 32               | 81           |
|                       |   |                                       | 33               | 108          |
|                       |   |                                       | 34               | 121.5        |
|                       |   |                                       | 35               | 135          |
|                       |   |                                       | 36               | 27           |
|                       |   |                                       | 37               | 54           |
|                       |   |                                       | 38               | 81           |
|                       |   |                                       | 39               | 108          |
|                       |   |                                       | 40               | 162          |
|                       |   |                                       | 41               | 216          |
|                       |   |                                       | 42               | 243          |
|                       |   |                                       | 43               | 270          |
|                       |   |                                       | 44               | 2(S)         |
|                       |   |                                       | 45               | 5.5(S)       |
|                       |   |                                       | 46               | 11(S)        |
| --txpwr               | -txpwr <1-30 dBm>   | Output power ranging from 1 to 30 dBm | Index            | Output power |
|                       |   |                                       | 1                | 1 dBm        |
|                       |   |                                       | 15               | 15 dBm       |
|                       |   |                                       | ...              | ...          |
|                       |   |                                       | 30               | 30 dBm       |

**Table 13-33 athtestcmd Frequently Used Options (cont.)**

| Command option | Example  | Description   | Parameter  |   |
|----------------|--|---|--|---|
| --txpattern    | <b>--txpattern</b><br><0: all zeros;<br>1: all ones;<br>2: repeating 10;<br>3: PN7;<br>4: PN9;<br>5: PN15> |   | Index  | Pattern   |
|                |  |   | 0  | All zeros   |
|                |  |   | 1  | All ones  |
|                |  |   | 2  | Repeating 10  |
|                |  |   | 3  | Sudo random 7<br>Repeat sudo<br>Random string<br>of size 7 (i.e<br>1527123<br>1527123 |
|                |  |   | 4  | Sudo random 9   |
|                |  |   | 5  | Sudo random 9   |
| --ani          | --ani  | Enable ANI. The ANI is disabled if this option is not specified | NO Parameter   |   |
| --scrambleroff | --scrambleroff   | Disable scrambler. The scrambler is enabled by default          | NO Parameter   |   |
| --shortguard   | --shortguard   | Transmit with shortguard interval                               | NO Parameter   |   |
| --mode         | --mode   | Set the TX mode/bandwidth                                       | ht40plus/ht40minus/ht20/vht20/<br>vht40plus/vht40minus/vht80_0/<br>vht80_1/vht80_2/vht80_3 |   |
| --rx           | --rx promis  | Receive TX99 packets  | promis   | Receive in promiscuous mode   |
|                |  |   | report   | Display tx99 packet info  |

#### 13.17.4.2 All athtestcmd TX options

```
--tx <frame/tx99/tx100/sine/off>
    --txfreq <Tx channel or freq(default 2412)>
    --txrate <rate index>
    --txpwr <0-30dBm, 0.5 dBm resolution; sine: 0-60, PCDAC value>
    --tgtpwr <target power>
    --pcdac <power control DAC>
    --gainIdx <tx gain table index>
    --dacGain <DAC gain>
    --forcedGain <power control DAC>
    --txantenna <1/2/0 (auto)>
    --txpktsz <pkt size, [32-1500](default 1500)>
    --txpattern <tx data pattern, 0: all zeros; 1: all ones; 2:  
repeating 10; 3: PN7; 4: PN9; 5: PN15
        --txchain (1:on chain 0, 2:on chain 1, 3:on both)
        --tpcm <forcedTgtPwr/forcedGainIdx/forcedGain/txpwr/tgtpwr> : Set  
transmit power control mode, by default tpcm is set to txpwr
```

```

--ani (Enable ANI. The ANI is disabled if this option is not
specified)
--paprd (Enable PAPRD. The PAPRD is disabled if this option is not
specified)
--stbc (Enable STBC. STBC is disabled if this option is not
specified)
--ldpc (Enable LDPC. LDPC is disabled if this option is not
specified)
--scrambleroff (Disable scrambler. The scrambler is enabled by
default)
--aifsn <AIFS slots num,[0-252](Used only under '--tx frame' mode)>
--shortguard (use short guard)
--mode <ht40plus/ht40minus/ht20/vht20/vht40plus/vht40minus/vht80_
0/vht80_1/vht80_2/vht80_3>
--bw <full/half/quarter>
--setlongpreamble <1/0>
--numpackets <number of packets to send 0-65535>
--tx sine --txfreq <Tx channel or freq(default 2412)>
--agg <number of aggregate packets>
--bssid
--srcmac <source MAC address>
--dstmac <destination MAC address>
--tx sine --txfreq <Tx channel or freq(default 2412)>

```

#### 13.17.4.3 All athtestcmd RX options

```

--rx <promis/filter/report>
--rxfreq <Rx channel or freq(default 2412)>
--rxantenna <1/2/0 (auto)>
--rxchain <1:chain 0, 2:chain 1, 3:both>
--mode <ht40plus/ht40minus>
--ack <Enable/Disable ACK, 0:disable; 1:enable, by default ACK is
enabled>
--bc <0:receive unicast frame, 1:receive broadcast frame (default)>
--lpl <0:LPL off(default), 1:reduced receive, 2:reduced search>

```

#### 13.17.4.4 All athtestcmd other options

```

--pm <wakeup/sleep/deepsleep>
--setmac <mac addr like 00:03:7f:be:ef:11>
--setbssid <mac addr like 00:03:7f:be:ef:11>
--SetAntSwitchTable <table1 in hex value> <table2 in hex value> (Set
table1=0 and table2=0 will restore the default AntSwitchTable)
--efusedump --start <start address> --end <end address>
--efusewrite --start <start address> --data <data> (could be one or
multiple data in quotation marks)
--otpwrite --data (could be one or multiple data in quotation marks)
--otpdump
--btmode <put DUT into BT mode>

```

### 13.17.5 athtestcmd Tx Rate Table

**Table 13-34 athtestcmd Tx Rate Table**

| Value     | Rate               |
|-----------|--------------------|
| <rate> 0  | 1 Mb               |
| <rate> 1  | 2 Mb               |
| <rate> 2  | 5.5 Mb             |
| <rate> 3  | 11 Mb              |
| <rate> 4  | 6 Mb               |
| <rate> 5  | 9 Mb               |
| <rate> 6  | 12 Mb              |
| <rate> 7  | 18 Mb              |
| <rate> 8  | 24 Mb              |
| <rate> 9  | 36 Mb              |
| <rate> 10 | 48 Mb              |
| <rate> 11 | 54 Mb              |
| <rate> 12 | HT20 MCS0 6.5 Mb   |
| <rate> 13 | HT20 MCS1 13 Mb    |
| <rate> 14 | HT20 MCS2 19.5 Mb  |
| <rate> 15 | HT20 MCS3 26 Mb    |
| <rate> 16 | HT20 MCS4 39 Mb    |
| <rate> 17 | HT20 MCS5 52 Mb    |
| <rate> 18 | HT20 MCS6 58.5 Mb  |
| <rate> 19 | HT20 MCS7 65 Mb    |
| <rate> 20 | HT20 MCS8 13 Mb    |
| <rate> 21 | HT20 MCS9 26 Mb    |
| <rate> 22 | HT20 MCS10 39 Mb   |
| <rate> 23 | HT20 MCS11 52 Mb   |
| <rate> 24 | HT20 MCS12 78 Mb   |
| <rate> 25 | HT20 MCS13 104 Mb  |
| <rate> 26 | HT20 MCS14 117 Mb  |
| <rate> 27 | HT20 MCS15 130 Mb  |
| <rate> 28 | HT20 MCS16 19.5 Mb |
| <rate> 29 | HT20 MCS17 39 Mb   |
| <rate> 30 | HT20 MCS18 58.5 Mb |
| <rate> 31 | HT20 MCS19 78 Mb   |
| <rate> 32 | HT20 MCS20 117 Mb  |
| <rate> 33 | HT20 MCS21 156 Mb  |

**Table 13-34 athtestcmd Tx Rate Table**

| <b>Value</b> | <b>Rate</b>           |
|--------------|-----------------------|
| <rate> 34    | HT20 MCS22 175.5 Mb   |
| <rate> 35    | HT20 MCS23 195 Mb     |
| <rate> 36    | HT40 MCS0 13.5 Mb     |
| <rate> 37    | HT40 MCS1 27.0 Mb     |
| <rate> 38    | HT40 MCS2 40.5 Mb     |
| <rate> 39    | HT40 MCS3 54 Mb       |
| <rate> 40    | HT40 MCS4 81 Mb       |
| <rate> 41    | HT40 MCS5 108 Mb      |
| <rate> 42    | HT40 MCS6 121.5 Mb    |
| <rate> 43    | HT40 MCS7 135 Mb      |
| <rate> 44    | HT40 MCS8 27 Mb       |
| <rate> 45    | HT40 MCS9 54 Mb       |
| <rate> 46    | HT40 MCS10 81 Mb      |
| <rate> 47    | HT40 MCS11 108 Mb     |
| <rate> 48    | HT40 MCS12 162 Mb     |
| <rate> 49    | HT40 MCS13 216 Mb     |
| <rate> 50    | HT40 MCS14 243 Mb     |
| <rate> 51    | HT40 MCS15 270 Mb     |
| <rate> 52    | HT40 MCS16 40.5 Mb    |
| <rate> 53    | HT40 MCS17 81 Mb      |
| <rate> 54    | HT40 MCS18 121.5 Mb   |
| <rate> 55    | HT40 MCS19 162 Mb     |
| <rate> 56    | HT40 MCS20 243 Mb     |
| <rate> 57    | HT40 MCS21 324 Mb     |
| <rate> 58    | HT40 MCS22 364.5 Mb   |
| <rate> 59    | HT40 MCS23 405 Mb     |
| <rate> 60    | VHT20 MCS0 S1 6.5 Mb  |
| <rate> 61    | VHT20 MCS1 S1 13 Mb   |
| <rate> 62    | VHT20 MCS2 S1 19.5 Mb |
| <rate> 63    | VHT20 MCS3 S1 26 Mb   |
| <rate> 64    | VHT20 MCS4 S1 39 Mb   |
| <rate> 65    | VHT20 MCS5 S1 52 Mb   |
| <rate> 66    | VHT20 MCS6 S1 58.5 Mb |
| <rate> 67    | VHT20 MCS7 S1 65 Mb   |
| <rate> 68    | VHT20 MCS8 S1 78 Mb   |
| <rate> 69    | VHT20 MCS9 S1 86 Mb   |

**Table 13-34 athtestcmd Tx Rate Table**

| <b>Value</b> | <b>Rate</b>            |
|--------------|------------------------|
| <rate> 70    | VHT20 MCS0 S2 13 Mb    |
| <rate> 71    | VHT20 MCS1 S2 26 Mb    |
| <rate> 72    | VHT20 MCS2 S2 39 Mb    |
| <rate> 73    | VHT20 MCS3 S2 52 Mb    |
| <rate> 74    | VHT20 MCS4 S2 78 Mb    |
| <rate> 75    | VHT20 MCS5 S2 104 Mb   |
| <rate> 76    | VHT20 MCS6 S2 117 Mb   |
| <rate> 77    | VHT20 MCS7 S2 130 Mb   |
| <rate> 78    | VHT20 MCS8 S2 156 Mb   |
| <rate> 79    | VHT20 MCS9 S2 173      |
| <rate> 80    | VHT20 MCS0 S3 19.5 Mb  |
| <rate> 81    | VHT20 MCS1 S3 39 Mb    |
| <rate> 82    | VHT20 MCS2 S3 58.5 Mb  |
| <rate> 83    | VHT20 MCS3 S3 78 Mb    |
| <rate> 84    | VHT20 MCS4 S3 117 Mb   |
| <rate> 85    | VHT20 MCS5 S3 156 Mb   |
| <rate> 86    | VHT20 MCS6 S3 175.5 Mb |
| <rate> 87    | VHT20 MCS7 S3 195 Mb   |
| <rate> 88    | VHT20 MCS8 S3 234 Mb   |
| <rate> 89    | VHT20 MCS9 S3 260 Mb   |
| <rate> 90    | VHT40 MCS0 S1 13.5 Mb  |
| <rate> 91    | VHT40 MCS1 S1 27 Mb    |
| <rate> 92    | VHT40 MCS2 S1 40.5 Mb  |
| <rate> 93    | VHT40 MCS3 S1 54 Mb    |
| <rate> 94    | VHT40 MCS4 S1 81 Mb    |
| <rate> 95    | VHT40 MCS5 S1 108 Mb   |
| <rate> 96    | VHT40 MCS6 S1 121.5 Mb |
| <rate> 97    | VHT40 MCS7 S1 135 Mb   |
| <rate> 98    | VHT40 MCS8 S1 162 Mb   |
| <rate> 99    | VHT40 MCS9 S1 180 Mb   |
| <rate> 100   | VHT40 MCS0 S2 27 Mb    |
| <rate> 101   | VHT40 MCS1 S2 54 Mb    |
| <rate> 102   | VHT40 MCS2 S2 81 Mb    |
| <rate> 103   | VHT40 MCS3 S2 108 Mb   |
| <rate> 104   | VHT40 MCS4 S2 162 Mb   |
| <rate> 105   | VHT40 MCS5 S2 216 Mb   |

**Table 13-34 athtestcmd Tx Rate Table**

| Value      | Rate                   |
|------------|------------------------|
| <rate> 106 | VHT40 MCS6 S2 243 Mb   |
| <rate> 107 | VHT40 MCS7 S2 270 Mb   |
| <rate> 108 | VHT40 MCS8 S2 324 Mb   |
| <rate> 109 | VHT40 MCS9 S2 360 Mb   |
| <rate> 110 | VHT40 MCS0 S3 40.5 Mb  |
| <rate> 111 | VHT40 MCS1 S3 81 Mb    |
| <rate> 112 | VHT40 MCS2 S3 121.5 Mb |
| <rate> 113 | VHT40 MCS3 S3 162 Mb   |
| <rate> 114 | VHT40 MCS4 S3 243 Mb   |
| <rate> 115 | VHT40 MCS5 S3 324 Mb   |
| <rate> 116 | VHT40 MCS6 S3 364.5 Mb |
| <rate> 117 | VHT40 MCS7 S3 405 Mb   |
| <rate> 118 | VHT40 MCS8 S3 486 Mb   |
| <rate> 119 | VHT40 MCS9 S3 540 Mb   |
| <rate> 120 | VHT80 MCS0 S1 29.3 Mb  |
| <rate> 121 | VHT80 MCS1 S1 58.5 Mb  |
| <rate> 122 | VHT80 MCS2 S1 87.8 Mb  |
| <rate> 123 | VHT80 MCS3 S1 117 Mb   |
| <rate> 124 | VHT80 MCS4 S1 175.5 Mb |
| <rate> 125 | VHT80 MCS5 S1 234 Mb   |
| <rate> 126 | VHT80 MCS6 S1 263.3 Mb |
| <rate> 127 | VHT80 MCS7 S1 292.5 Mb |
| <rate> 128 | VHT80 MCS8 S1 351 Mb   |
| <rate> 129 | VHT80 MCS9 S1 390 Mb   |
| <rate> 130 | VHT80 MCS0 S2 58.5 Mb  |
| <rate> 131 | VHT80 MCS1 S2 117 Mb   |
| <rate> 132 | VHT80 MCS2 S2 175.5 Mb |
| <rate> 133 | VHT80 MCS3 S2 234 Mb   |
| <rate> 134 | VHT80 MCS4 S2 351 Mb   |
| <rate> 135 | VHT80 MCS5 S2 468 Mb   |
| <rate> 136 | VHT80 MCS6 S2 526.5 Mb |
| <rate> 137 | VHT80 MCS7 S2 585 Mb   |
| <rate> 138 | VHT80 MCS8 S2 702 Mb   |
| <rate> 139 | VHT80 MCS9 S2 780 Mb   |
| <rate> 140 | VHT80 MCS0 S3 87.8 Mb  |
| <rate> 141 | VHT80 MCS1 S3 175.5 Mb |

**Table 13-34 athtestcmd Tx Rate Table**

| Value      | Rate                   |
|------------|------------------------|
| <rate> 142 | VHT80 MCS2 S3 263.3 Mb |
| <rate> 143 | VHT80 MCS3 S3 351 Mb   |
| <rate> 144 | VHT80 MCS4 S3 526.5 Mb |
| <rate> 145 | VHT80 MCS5 S3 702 Mb   |
| <rate> 146 | VHT80 MCS6 S3 789.8 Mb |
| <rate> 147 | VHT80 MCS7 S3 877.5 Mb |
| <rate> 148 | VHT80 MCS8 S3 1053 Mb  |
| <rate> 149 | VHT80 MCS9 S3 1170 Mb  |
| <rate> 150 | 2(S) Mb                |
| <rate> 151 | 5.5(S) Mb              |
| <rate> 152 | 11(S) Mb               |

### 13.17.5.1 athtestcmd example

#### RX on channel 149

```
athtestcmd -i wifi1 --rx promis --rxfreq 149
athtestcmd -i wifi1 --rx report
```

#### TX at channel 149, txchain 1, rate 12, and in VHT80 mode

```
athtestcmd -i wifi1 --tx tx99 --txfreq 149 --txchain 1 --txrate 12 --
mode vht80_0
```

#### TX off

```
athtestcmd --tx off
```

### 13.17.6 Testing with tx99tool and athtestcmd

#### 13.17.6.1 tx99tool

##### AP setup example

```
apdown
cfg -x
cfg -a AP_IPADDR=15.15.15.20
cfg -a AP_NETMASK=255.255.255.0
cfg -a AP_STARTMODE=dual
cfg -a AP_PRIMARY_CH_2=149
cfg -a AP_CHMODE_2=11NAHT40PLUS
cfg -a AP_MODE_2=ap
cfg -a AP_SSID_2="tuliplily5G"
cfg -a AP_SECMODE_2=None
cfg -a AP_PRIMARY_CH=9
cfg -a AP_CHMODE=11NGHT40PLUS
```

```
cfg -a AP_MODE=ap
cfg -a AP_SSID="tuliplily2G"
cfg -a AP_SECMODE=None
cfg -a AP_ENABLE_TX99=y
cfg -c
apup
```

### 13.17.6.2 Run tx99tool example

#### Setup and start TX

```
tx99tool wifi0 set freq 1 bandwidth 0 htext 0
tx99tool wifi0 set rate 6000
tx99tool wifi0 set txmode 6
tx99tool wifi0 set txchain 1
tx99tool wifi0 set testmode 0
tx99tool wifi0 set pwr 20
tx99tool wifi0 start
```

#### Stop TX

```
tx99tool wifi0 stop
```

### 13.17.6.3 athtestcmd

#### AP setup example

2 APs are needed, 1 for TX, another for RX, the following are the AP setups.

#### AP1 RX

```
cfg -a AP_MODE=ap
cfg -a AP_IPADDR=192.168.2.101
cfg -a AP_NETMASK=255.255.255.0
cfg -a AP_STARTMODE=dual
cfg -a AP_PRIMARY_CH=6
cfg -a AP_CHMODE=11NGHT40PLUS
cfg -a AP_SSID="NZHAO_RX_DEBUG_2G"
cfg -a AP_SECMODE=None
cfg -a AP_MODE_2=ap
cfg -a AP_PRIMARY_CH_2=149
cfg -a AP_CHMODE_2=11NAHT40PLUS
cfg -a AP_SSID_2="NZHAO_RX_DEBUG_5G"
cfg -a AP_SECMODE_2=None
cfg -a AP_ENABLE_TX99=y
cfg -c
apup
```

#### AP2 TX

```
cfg -a AP_MODE=ap
cfg -a AP_IPADDR=192.168.2.102
cfg -a AP_NETMASK=255.255.255.0
cfg -a AP_STARTMODE=dual
cfg -a AP_PRIMARY_CH=6
cfg -a AP_CHMODE=11NGHT40PLUS
```

```

cfg -a AP_SSID="NZHAO_TX_DEBUG_2G"
cfg -a AP_SECMODE=NONE
cfg -a AP_MODE_2=ap
cfg -a AP_PRIMARY_CH_2=149
cfg -a AP_CHMODE_2=11NAHT40PLUS
cfg -a AP_SSID_2="NZHAO_TX_DEBUG_5G"
cfg -a AP_SECMODE_2=NONE
cfg -a AP_ENABLE_TX99=y
cfg -c
apup

```

### Run athtestcmd example

#### On AP2, Setup and start TX

```
athtestcmd -i wifi1 --tx tx99 --txfreq 149 --txchain 1 --txrate 120
--mode vht80_0
```

#### On AP1, Start RX

```
athtestcmd -i wifi1 --rx promis --rxfreq 149
athtestcmd -i wifi1 --rx report
```

#### Sample RX results

```

~ # athtestcmd -i wifi1 --rx promis --rxfreq 149 --mode vht80_0
Its cont Rx promis mode
Response status 0
~ # athtestcmd -i wifi1 --rx report
Its cont Rx report mode
totalpkt 3839
rssInDBm 60
crcErrPkt 0
secErrPkt 0
VHT80 MCS0 S1 29.3 Mb .. rateCnt 3839

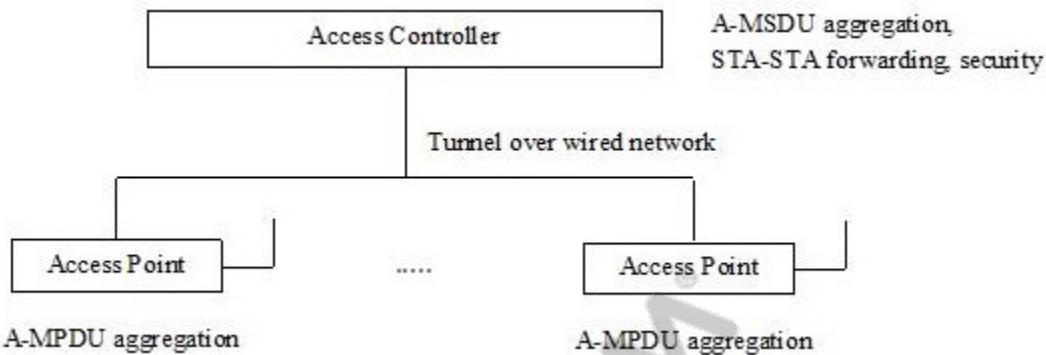
```

Here RSSI, mode, rate info are printed with the ‘report’ command.

## 13.18 Raw Mode Tx/Rx

### 13.18.1 Introduction

The raw mode is an encapsulation/decapsulation configuration wherein the Qualcomm Technologies host driver exchanges raw 802.11 frames with the hardware, instead of the default Ethernet frames. This allows external entities such as an Access Controller (AC) to exchange raw frames with Qualcomm Technologies based thin APs over say, a GRE tunnel; see [Figure 13-23](#). The most important use case for this is centralized handling of encryption/decryption by the AC instead of the APs, allowing easier certification for military needs. The keys will reside on the AC and not on the individual APs. The raw mode also offers other potential benefits such as centralized decision making.



**Figure 13-23 Access Points Controlled by Access Controller**

This feature is currently supported only for AR900B on 10.4. This section discusses the scope and design of the host side implementation.

**NOTE** The reader should be familiar with the default host side data path. This section mostly focuses on those aspects which differ from this.

## 13.18.2 Scope

### 13.18.2.1 Requirements

**NOTE** The requirements are currently driven by specific customer requirements.

- The host WLAN driver must support Tx and Rx of the raw 802.11 MPDUs.
- Only AP mode support is required; WDS support is not required.
- The Tx and Rx of an A-MSDU bearing raw MPDUs up to the maximum size, stipulated by the 802.11ac standard (11,454 bytes) should be supported.

Based on this, the DMA scatter/gather should be implemented in the host WLAN driver.

- The AC should carry out encryption/decryption of the MPDUs, and the host driver should provide the facility to pass through the encrypted raw MPDUs as-is.
- The A-MSDU aggregation of the raw frames will be carried out by the AC, while the A-MPDU aggregation will be carried out by the AP.

**NOTE** The host driver has no role to play in the A-MPDU aggregation. This is handled between the firmware and the hardware. This requirement is listed for completeness.

- During Tx, the AP should suitably modify some of the 802.11 header fields in the raw MPDU provided by the AC, as per protocol state factors that vary at actual transmission time (for example, retry bit, power save related bits).

**NOTE** The host driver has no role to play in this. This is handled between the firmware and the hardware. This requirement is listed for completeness.

- Usage of Ethernet/Native Wi-Fi or raw mode should be runtime configurable on a per-VAP basis, separately for Tx and Rx.
- It should be possible to have mixed VAPs, i.e., some VAPs configured for the raw mode encapsulation/decapsulation, some for the Ethernet encapsulation/decapsulation.
- If a VAP is in the Ethernet mode, differences related to the raw mode does not apply. Thus, both the A-MSDU and A-MPDU aggregation have to be carried out by the AP. Security will be handled entirely by the AP.
- It should be possible for the raw MPDUs to be exchanged between the host WLAN driver and an external interfacing module. Similarly, it should be possible for the interfacing code to exchange Ethernet frames using either a tunnel with an external entity like an AC, or a bridge with further layers.

The external interfacing module, AC, and the tunneling/bridging are not part of the Qualcomm Technologies deliverables. They have to be handled separately by the customers as per their end designs.

However, the host WLAN driver should implement skb chaining using the next pointer (for large MPDUs exceeding ~1.5 K). The Linux frag\_list mechanism is not used, only next pointer based chaining and termination of the chain using NULL pointer is implemented, as per current customer needs; see [Section 13.18.3.2](#). For future requirements that need the Linux frag\_list mechanism, it is relatively simple to add converter code at the WLAN driver entry/exit points.

- A simple simulation should be added for Qualcomm Technologies internal testing purposes, since it is currently not possible to interface with an AC. This simulation converts the Ethernet Type II to and from the raw 802.11 MPDUs, at offload driver entry/exit points. Due to this, the AP can exchange Ethernet Type II frames with external hosts connected to it through the Ethernet cables, while exchanging raw 802.11 frames along the internal data paths.

### 13.18.2.2 Nonapplicable functionality

#### Data path alternatives

##### Tx

Only the Tx fast path with the cached header host data path is supported. The Tx non-fast path is not supported (it is not supported by the rest of the host WLAN driver as well, as of driver version 10.4). The older Tx fast path without cached header is meant for features which cannot work with cached header, and is not covered; it is selected per Tx packet only for those particular features.

##### Rx

The native Wi-Fi based Rx IP alignment WAR required for AR9888 is not required for AR900B and hence the current implementation does not factor this in.

## STA-to-STA forwarding and re-injection

The Intra BSS STA-to-STA communication is currently not applicable since this involves AC forwarding. This also applies to multicast re-injection.

### 13.18.2.3 Limitations

This section outlines known restrictions. Other restrictions might apply depending on the details of integration with end customer platforms and other factors.

#### Simulation

This simulation is aimed specifically at aiding Qualcomm Technologies internal testing of raw mode without an AC (and hence not full-fledged).

- It does not handle IBSS, P2P, 4-address frames in the Tx direction.
- It does not handle WAPI. It cannot handle encryption/decryption and the hardware needs to handle this prior to the simulation. It is tested using PSK security only in a combination, where as Tx uses 802.3 and Rx uses the raw mode. It is suggested to set Rx into raw mode after association.
- It only handles plain Ethernet II frames and does not handle variations such as tagged frames.
- The A-MSDU handling does not carry out certain types of verification such as the check of DA/SA/TID coherence, since the main aim is to send/receive large frames and such coherence verification can be done on the AC. The A-MSDU handling may not be able to process certain other unexpected protocol violations by the peer. It does attempt to work around some scenarios but the coverage might not be complete. Usage of uncertified/buggy STA drivers could lead to undefined behavior and instability in the simulation (for example, certain pre-releases or non-Wi-Fi certified 802.11ac products).
- The Tx side A-MSDU generation is simple and doesn't adapt dynamically – this can be done in AC (and is done in firmware for Ethernet II). Hence, there is a possibility of packet loss if A-MSDU generation is enabled in the simulation.
- The 802.11 Rx defragmentation is not interfaced with the simulation module and is not separately testable under the raw mode configuration. It is to be directly connected to the interfacing module in end customer systems. See [Section](#) for details.

**NOTE** The driver has not been tested (and is not currently testable by Qualcomm Technologies) with simulation code disabled.

#### skb exchange with OS

The creation and consumption of the raw frame bearing skbs (whether single or chained) starts and terminates at the simulation, and doesn't involve the rest of the OS. An external interfacing module would be required for exchanges beyond the host WLAN driver (see [Section 13.18.2.1](#)).

#### Security

For security, only the mode in which the AC carries out all security-related operations is supported (with a minor code change required in end systems; see [Table 13-35](#)). The host driver does not

have the capability to carry out security-related processing for the raw mode, though there is a limited Rx side non-WAPI PSK-based functionality for Qualcomm Technologies internal testing purposes. While the hardware can be configured to carry out encryption/decryption, capabilities such as 802.1X functionality are not available in the host driver currently for raw mode.

**NOTE** 802.1X should work if the AC handles it.

### 802.11 Rx defragmentation

The 802.11 Rx defragmentation already uses the raw mode (even for non-raw cases) over a separate slow path. At the end of the path, the raw frames are decapsulated to the Ethernet. No exceptions are currently made for the raw mode. The interfacing module for the raw mode (see [Section 13.18.2.1](#)) should skip this decapsulation. Depending on the end customer needs, the defragmentation process too could be skipped by the customer-designed interfacing module and MPDU fragments sent as-is to the AC.

### Statistics

The packet count statistics are not covered since these are best processed at the AC (given the limitations like encryption).

#### 13.18.2.4 Assumptions

- Since AR900B supports up to six fragments per SDU, a simple default buffer coalescing scheme is to be used for a higher number of fragments to support 802.11ac (with a maximum MPDU size of 11,454 bytes). The scheme copies the sixth fragment onwards into a newly allocated skb.

**NOTE** Interfacing module/higher layers for end implementations should restrict the number of fragments to six, using means like 6x2 K sized jumbo frames or NSS copy.

- TSO and nr\_frags based scatter/gather will not be used along with the raw mode since these are not currently part of the requirements.

#### 13.18.3 Design

##### 13.18.3.1 Simulation

The simulation code has been added at the driver entry and exit points under the compilation flag, *QCA\_SUPPORT\_RAWMODE\_PKT\_SIMULATION*. It acts only on those VAPs which are configured for the raw Tx/Rx.

##### Tx

The Tx side code converts incoming Ethernet II frames into 802.11 MPDUs. Additionally, if the incoming frames are IP fragments, it combines up to two IP fragments into an A-MSDU, with each MSDU corresponding to one IP fragment. Chaining is used here to exercise scatter/gather. Note testing with higher number of fragments has been carried out with up to eight fragments.

The Tx side cannot handle crypto frames, including un-encrypted ones with only the requisite fields.

## Rx

The Rx side code converts incoming 802.11 MPDUs into Ethernet II frames. If the MPDUs are A-MSDU bearing ones, these are parsed to decapsulate each MSDU into a separate Ethernet II frame. The first skb bearing the MPDU is used inplace to bear the first Ethernet II frame, but new skbs are allocated for the remaining Ethernet II frames.

Additionally, the Rx side code strips off the crypto headers and trailers if the decrypted 802.11 MPDUs arrive. Note that the Rx side code is incapable of handling the decryption process itself (and this is not a requirement). It can only handle hardware decrypted MPDUs.

### 13.18.3.2 Scatter/gather support

The scatter/gather support is added in order to enable large frame Tx/Rx (towards A-MSDU handling).

The fragments flowing through the WLAN driver must not be in Linux *frag\_list* format, since the current customer requirements need a plain next pointer based approach and a trim cannot be done on the first fragment (which in some cases might be needed for removing extraneous bytes).

The next based chaining mechanism is used within the driver. At the boundary with the OS (such as *hard\_start\_xmit()* or *netif\_rx()*), a complete solution that requires the *frag\_list* mechanism involves converting between the *frag\_list* and that used in the driver. See section [Section 13.18.2.1](#).

### 13.18.3.3 Tx side

The changes are carried out for the fast path with the cached header.

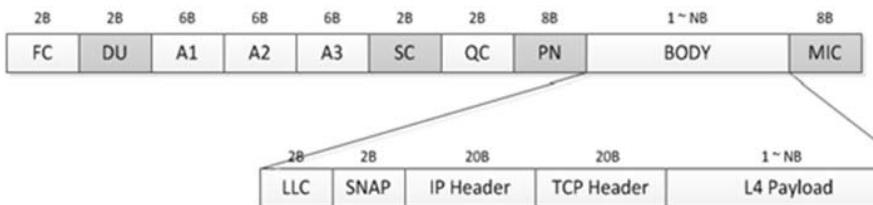
- When a VDEV is configured for the raw mode Tx, the cached header which is used to populate the HTT descriptor for every packet in *ol\_tx\_ll\_cachedhdr\_prep()*, is modified at the configuration time as listed in [Table 13-35](#).

**Table 13-35 Packet type and subtype configuration**

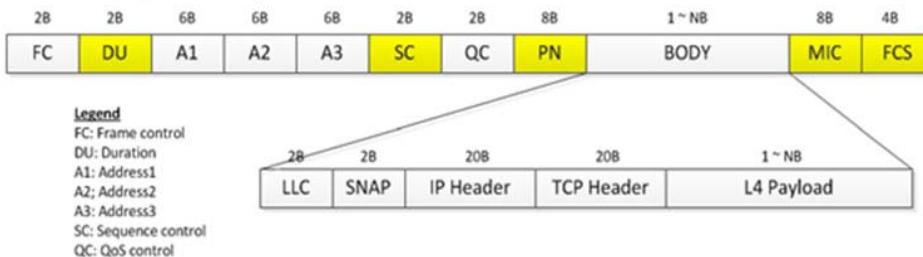
| Field  | Setting      |     |                                   |
|--|--------------|-----|-----------------------------------|
| <i>pkt_type</i>  | 0 (Raw mode) |     |                                   |
| <i>pkt_subtype</i>   | Bit [0]      | 0x1 | 802.11 MAC header is present.     |
|  | Bit [1]      | 0x1 | Does not allow aggregation.       |
|  | Bit [2]      | 0x0 | Performs encryption.<br><b>OR</b> |
|  |              | 0x1 | Does not perform encryption.      |
| <p><b>NOTE</b> Qualcomm Technologies currently keeps this at 0x0 for all cases because the reference code will not be tested against an external AC that can do the encryption.</p> <p>End systems can configure this appropriately.</p> |              |     |                                   |

- The AC (or simulation) provides a raw MPDU to the WLAN driver for transmission. The following apply:
  - Some fields can be kept blank as given in [Figure 13-24](#).
  - Some fields such as retry and power save related bits can be updated by firmware/hardware as per protocol state factors that vary at the actual transmission time.

Raw encap: Input – 802.11 frame  
with the field in grey empty:



Raw encap: Output – 802.11 frame  
with the field in yellow filled by HW

**Figure 13-24 Tx Raw frame**

- During the transmission of each MPDU, *osif\_ll\_vap\_hardstart()* calls *OL\_TX\_LL\_UMAC\_RAW\_PROCESS()*, which carries out an *skb\_unshare()* and directly calls *OL\_TX\_LL\_WRAPPER()* thus skipping procedures in *osif\_ll\_vap\_hardstart()* that do not apply, such as VLAN WAR, Extender AP functionality, VoW debug, Multicast to Unicast conversion, TSO, and so on.
- If no additional fragments are indicated in the skbs coming in from the network stack, the usual process is followed in *ol\_tx\_ll\_cachedhdr\_prep()*.
- If additional fragments are indicated in the incoming skb, following steps are carried out in the *OL\_TX\_PREPARE\_RAW\_DESC\_CHDHDR()* called from *ol\_tx\_ll\_cachedhdr\_prep()*:
  - In case the total number of fragments exceed six, call *ol\_raw\_tx\_coalesce\_nbufs()* to coalesce buffers. It copies the sixth fragment onwards into a newly allocated skb.
  - For efficiency, the restriction – fragments 1–5 should have at least 1500 bytes each, is enforced. This is done to demand a reasonably sized new skb from the OS. The total length of fragments should not cross the 802.11 maximum MPDU size, i.e., 11,454 bytes.
  - The coalescing code can be disabled at the compile time.
  - The DMA maps each skb separately (excluding the first one). The physical address is stored in the private area of each skb.
  - Set the physical address of each fragment into the corresponding entry in the MSDU Link Extension descriptor. In addition, set the length of each fragment in the descriptor. See [Figure 13-25](#).
  - Set the length in the HTT descriptor to the total size covering all the fragments (not just the length of the first fragment).
- Upon Tx completion, if the skb has next pointer set (currently, only for raw mode), unmap the fragments following the main skb. Unmapping of the main skb is carried out in current code as usual.

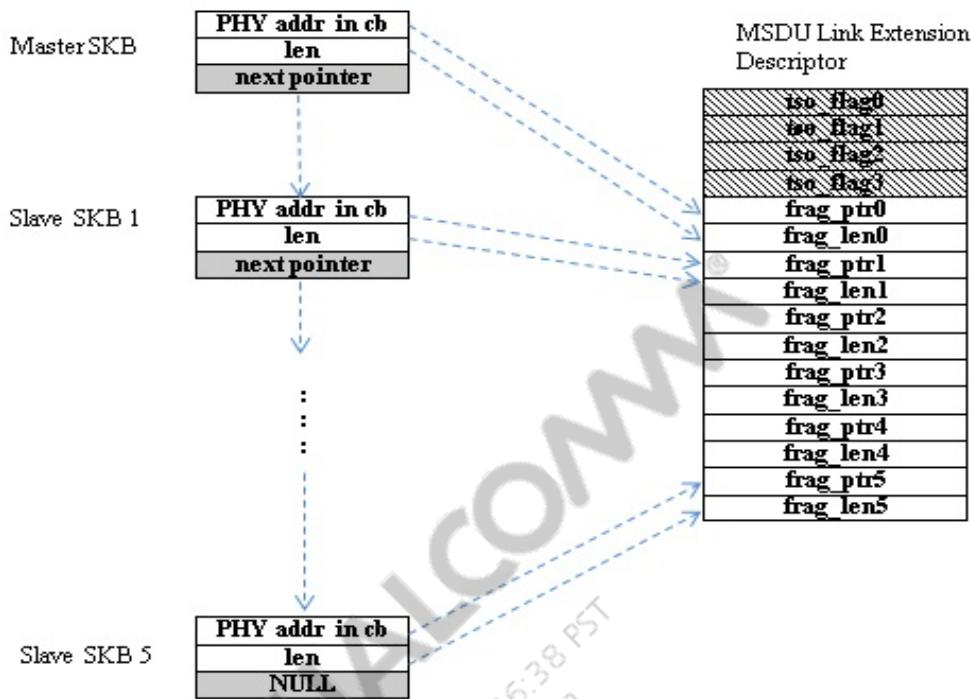


Figure 13-25 Tx side Mapping to MSDU Link Extension Descriptor

### 13.18.3.4 Rx side

**NOTE** Since the Rx path can group fragments belonging to multiple MPDUs, two 1-bit flags have been introduced into the skb private area. One flag indicates the first fragment skb and the other flag indicates the last. These can be examined in a loop at the driver boundary to group fragments belonging to the same MPDU.

- Rx skbs sized ~1500 are queued as usual.
- The field *ring2\_more\_count* in the Rx descriptor for first fragment indicates the number of additional fragments; extra fragments will not contain the Rx descriptor on top of the data. Each fragment from the hardware goes into a separate skb.
- On receiving an Rx indication, examine *ring2\_more\_count*:
  - If fragments are not present, proceed as usual.
  - If fragments are present, check for an FCS error. This error indicates that this is not a valid set of fragments. Ignore the fragments since they will ultimately be discarded (this check may not be strictly necessary since the discard logic is present elsewhere, but is put in as a precaution).
  - If there are no FCS errors, these are valid fragments. Set a bit in the skb private area of the first skb to indicate that it is the first in the chain, and set a bit for the last skb to indicate that it is the last.

- While passing the skbs upwards, forwarding checks in *ol\_rx\_fwd\_check()* are skipped since forwarding is not handled (the encryption keys would reside on the AC). However, if the code is extended to work for AR9888, requisite exceptions for mixed VAP implementations can be added by customers as required.
- Since Rx descriptor is available only for the master fragment, care should be taken not to try and access the descriptor for slave fragments. Besides, the descriptor for the master applies to the slaves. Hence in *ol\_rx\_pn\_check\_base()*, the function *OL\_RX\_MPDU\_LIST\_NEXT\_RAW()* is used to find the skb corresponding to the next MPDU instead of the regular *ol\_rx\_mpdu\_list\_next()*, if the VDEV is configured for raw Rx. This alternate function takes care of this precaution.
- Finally, *ol\_rx\_deliver()* calls *OL\_RX\_DELIVER\_RAW()* to deliver raw frames. If the code is extended to work for AR9888, the requisite exceptions for mixed VAP implementations can be added by customers. *OL\_RX\_DELIVER\_RAW()* mainly does the following:
  - If the firmware indicates that the MPDU is to be discarded, frees all skbs belonging to that MPDU.
  - Adds the skbs that have not been discarded to a linked list of skbs.
  - If the simulation is enabled, passes the raw skbs to the simulation to convert into Ethernet.
  - Minor: Calls applicable LED functionality.
  - Passes the linked list of skbs to the OS.

## 13.19 Customized neighbor report

This section describes the functionality to generate customized neighborhood report (NR) based on the SSID of NR request from STA. A response is sent only for the NR information that matches SSID in NR request from STA. If no SSID is presented in the NR request, AP filters the current SSID with which the STA associates and responds to STA. NR response must include NR information collected from all the radios with same SSID.

For example, if the NR response is for all three radios in a tri-radio, the scan table of each VAP of each radio is iterated, NR information is populated, and this derived NR statistic is sent to STA.

If shared flag is set and if no SSID presented in the NR request, then AP filters with current SSID with which the STA is associated and responds to STA. The maximum NR response packet length check is implemented, based on the maximum management frame length (*MAX\_TX\_RX\_PACKET\_SIZE*) to avoid packet boundary overflow. NR response fills until the maximum packet length size.

Use the *iwpriv athN nrshareflag* FLAG command to specify how the NR flag is shared between the wifi interfaces (that is, the two or three radios for which the NR response applies).

Bit [0~2] indicates the scan table of the corresponding radio (wifi0 ~ wifi2); denotes whether it is shared or not.

0, 1, 2, 4: Not shared.

3: Shared between wifi0 and wifi1

- 5: Shared between wifi0 and wifi2
- 6: Shared between wifi1 and wifi2
- 7: Shared cross wifi0, wifi1 and wifi2
- >7 && < 0xff Invalid. Input is refused
- : 0xff Disable the FR functionality

Use the `iwpriv athN get_nrshareflag` command to retrieve the current NR flag.

| Parameter                    | Format                                    | Description   |
|------------------------------|---|---|
| <code>nrshareflag</code>     | <code>iwpriv athN nrshareflag FLAG</code> | Bit [0~2] indicates the scan table of the corresponding radio (wifi0 ~ wifi2); denotes whether it is shared or not.<br>0, 1, 2, 4: Not shared.<br>3: Shared between wifi0 and wifi1<br>5: Shared between wifi0 and wifi2<br>6: Shared between wifi1 and wifi2<br>7: Shared cross wifi0, wifi1 and wifi2<br>>7 && < 0xff Invalid. Input will be refused<br>: 0xff Disable the FR functionality |
| <code>get_neshareflag</code> | <code>iwpriv athN get_nrshareflag</code>  | Get current NR share flag.  |

## 13.20 Mirror Tx packets from AP VAP to a monitor VAP in direct attach radios

The capability for mirroring the Tx packets from AP VAP to the monitor VAP of a direct attach radio is introduced. This functionality to capture the Tx packets of an AP VAP in a direct attach radio is available, only if the monitor mode is enabled in the same radio.

Typically, when monitor mode is enabled in a radio, Rx packets from other nearby APs are captured. If an AP VAP is set up along with the monitor VAP, the Tx packets of AP are not captured, which does not comply with the sniffer use case requirements. The sniffer use case requires both TX and RX packets to be delivered to the NW stack from driver.

Before the implementation of this functionality, in monitor mode configuration, only Rx packets can be captured if the VAP is configured in monitor mode. If an AP VAP is configured in the same radio as a monitor VAP, the Tx packets from the AP VAP cannot be captured in the monitor VAP.

The following design enhancements have been made to enable the mirroring of Tx packets:

1. Delivery of RX packets is supported through monitor mode support and complies with the sniffer use case requirements.

2. For Tx packets, the following operational changes apply:
  - a. For data packets, UMAC driver gets Tx completion of a packet from HW upon successful transmission or after the max retries have been exceeded.
  - b. When Tx completion is received for a packet, radiotap header is added for packet through `ieee80211_add_tx_radiotap_header()`
  - c. After adding the radiotap header, instead of freeing the buffer, the packet is delivered to the network stack using `netif_rx()` function of the kernel with the additional radiotap header information.
  - d. By the architecture, management packets, control packets, and self-generated packets of Tx are not available to be provided for sniffer usage.
3. Tx and Rx packets are interleaved because they are received and processed by different context, but the TSF information provided per packet allows the upper layers to order the packets
4. Configuration option is provided to enable/disable of TX/RX packet capture.

Before you configure the capability to mirror Tx packets sent from the AP VAP to the monitor VAP, configure regular DATA VAPs (such as ath0, ath1, and ath2) and configure the monitor VAP (such as athx) on the same radio.

Enable “tx\_capture\_da” support on the monitor VAP. Enter the `iwpriv athx tx_capture_da 1` command to enable this feature in DA mode. When interception of Tx packets or monitoring of Tx packets is enabled, Tx packets of all data VAPs are mirrored to the monitor VAP. Besides Tx packets, the monitor VAP also captures Rx packets in promiscuous mode.

If a user starts capturing the packets using the Tcpdump utility and then saves them in a .pcap file for viewing it later using a Wireshark tool, captured packets can be viewed on a remote Linux PC running Wireshark tool as and when packets are being received in network stack of AP.

## 13.21 Tx data packet captures in OL mode

Enable “tx\_capture” support on the monitor VAP. Enter the `iwpriv athx tx_capture 1` command to enable this feature in OL mode. When interception of Tx packets or monitoring of Tx packets is enabled, Tx packets of all data VAPs are mirrored to the monitor VAP. Besides Tx packets, the monitor VAP also captures Rx packets in promiscuous mode.

In association, dissociation and reassociation responses, and authentication and reauthentication responses, the mgmt completion handler gets the completion message with the PPDU\_ID contained in it. Examine the frame type to confirm that the received message frame is the Deauth response frame. The pktlog TX\_STAT message carries an event with the metadata information and the same PPDU\_ID.

- Association Response: A single VAP that functions as the DUT and STA associates with the DUT
- Reassociation Response: The STA that attempts to reassociate with the AP after the STA is moved to a place where the signal from the AP is low resulting in disassociation and placed back in the original position.

- Probe Response: Bring down the vap and bring it up again; The STA keeps sending probe requests
- Beacon: The VAP sends beacon frames at periodic intervals
- Disassociation Response: Disconnect the STA from the AP. The STA sends the disconnect request. The AP will send the disassoc response.
- Authentication Response: When the STA tries to connect to the DUT, it sends authentication request and the DUT sends Authentication Response frames.
- Deauthentication Response: The STA sends the deauth request and the AP sends the deauth response.

With ping packets sent between the STA and VAP, and iperf UDP and TCP tests run, the tx completion handler gets the completion message along with the PPDU\_ID.

The pktlog TX\_STAT message carries an event with the metadata information and the same PPDU\_ID

## 13.22 Transmission of MBO Reason Code attribute in BTM Request frames

AP transmits MBO Reason Code Attribute in BTM Request containing appropriate MBO Reason Code. AP parses MBO Reject Reason Code sent in BTM Response by STA if the BTM Request is rejected by it. If a BTM Request made by AP is rejected by STA with a valid MBO Reject Reason Code then that transaction has to be marked as “Best Effort” and spared from penalty.

If a BTM Request made by AP with MBO Reason Code 4 (LoadBalancing), and all or majority of STAs reject the request, then AP does not start “BTMSteeringProhibitShortTime” for these STAs and immediately sends another BTM Requests with Disassoc Imminent bit set and Disassoc Timer as 1 second to the STAs.

AP keeps track of such STAs that are forced to steer and bring them back to previous band as soon as the band does not become overloaded without waiting for “BTMSteerProhibitShortTime”. If any of the forced steered STAs is steered again for any other reason, then that STA no longer remains forced steered one.

### MBO Transition Reason Code Attribute

| Field                         | Size (Octets)      | Value (Hex) | Description   |  |  |
|-------------------------------|--------------------|-------------|---|--|--|
| Attribute ID                  | 1                  | 0x06        | Identifies the type of QCN Attribute.   |  |  |
| Attribute Length              | 1                  | 0x1         | Length of the following field of the attribute in octets.   |  |  |
| Transition Reason Code        | 1                  | Variable    | As specified in Transition Reason Codes table, identify the reason for this transition recommendation |  |  |
| <b>Transition Reason Code</b> | <b>Description</b> |             |   |  |  |
| 0                             | Unspecified        |             |   |  |  |

|        |  |  |
|--------|--|--|
| 1      | Excessive frame loss rate  |  |
| 2      | Excessive delay for current traffic stream   |  |
| 3      | Insufficient bandwidth for current traffic stream  |  |
| 4      | Load balancing   |  |
| 5      | Low RSSI   |  |
| 6      | Received excessive number of retransmissions   |  |
| 7      | High interference  |  |
| 8      | <p>Gray zone</p> <p>Imbalance between the PHY operating margin in the downlink direction compared to the uplink direction can result in a "gray" zone of coverage in which MBO STAs can become "stalled" in certain states such as:</p> <ul style="list-style-type: none"> <li>Mobile is 802.11 authenticated, but not associated</li> <li>Mobile is 802.11 associated, but not EAP authenticated</li> <li>Mobile is unable to obtain an IP address through DHCP</li> <li>Mobile is unable to perform name resolution through DNS</li> </ul> <p>If the serving MBO AP can detect that an MBO STA is in a gray zone, it should try to have the device transition.</p> |  |
| 9      | Transitioning to a Premium AP  |  |
| 10-255 | Reserved   |  |

### MBO Transition Reject Reason Code Attribute

| Field                            | Size (Octets)   | Value (Hex) | Description   |
|----------------------------------|---|-------------|---|
| Attribute ID                     | 1   | 0x07        | Identifies the type of MBO Attribute.   |
| Attribute Length                 | 1   | 0x1         | Length of the following field of the attribute in octets.   |
| Transition Rejection Reason Code | 1   | Variable    | As specified in Transition Rejection Reason Codes table, identify the reason that the MBO STA rejected this transition recommendation |
| Transition Rejection Reason Code | <i>Description</i>  |             |   |
| 0                                | Unspecified   |             |   |
| 1                                | Excessive frame loss rate expected by the MBO STA if it transitions to the suggested candidate BSS(s) in the BTM Request frame                            |             |   |
| 2                                | Excessive delay for current traffic stream would be incurred by BSS transition at this time   |             |   |
| 3                                | Insufficient QoS capacity for current traffic stream expected by the MBO STA if it transitions to the suggested candidate BSS(s) in the BTM Request frame |             |   |
| 4                                | Low RSSI in frames being received by the MBO STA from to the suggested candidate BSS(s) in the BTM Request frame  |             |   |
| 5                                | High interference expected by the MBO STA if it transitions to the suggested candidate BSS(s) in the BTM Request frame                                    |             |   |

|       |  |
|-------|--|
| 6     | Service Availability – the MBO STA expects that services it needs which are available at its serving AP will not be available if it transitions to the suggested candidate BSS(s) in the BTM Request frame |
| 7-255 | Reserved   |

### 13.22.1 Implementation of MBO Reason Code

Implementation of MBO Reason Code is performed in the following four stages:

1. The first stage implements passing the reason code from LBD to WLAN Driver through `dbgreq` ioctl.
2. The second stage implements inclusion of MBO Reason Code in the BTM request by WLAN driver.
3. The third stage implements sending MBO Reject Code in BTM Response from WLAN Driver to LBD.
4. The fourth stage implements handling of the exception case of STAs rejecting BTM Request originating due to load balancing. This last stage implements the force steering of such STAs and back steering them to original channel.

#### Code changes in lbd for adding MBO reason code

`steerexecImplCmnSteerBTM` : This function sends the BTM Request to Wifi Driver through `wlanif` module API “`wlanif_sendBTMRequest()`”. `state` variable in this function has the information about reason, which is internal to `steerexec` module. A function is written as “`steerReason = steerexecImplCmnSteerReasonToMboReason(state->reason)`”, which maps `state->reason` to MBO reason Code.

Added an input argument in “`wlanif_sendBTMRequest()`” to pass `steerReason`. “`wlanif_sendBTMRequest`” calls another function “`wlanifBSteerControlSendBTMRequest()`” which finally passes the BTM Request to Driver. An input argument is added to pass `steerReason` to this function.

In the function “`wlanifBSteerControlSendBTMRequest()`”, the BTM request structure is prepared and passed from LBD to Driver. The structure is called “`ieee80211_bstm_reqinfo_target`” which is defined in `qca-wifi-10.4`. A field is added in this structure called “`trans_reason`” to contain `steerReason` too. This function then issues calls a function `wlanifBSteerControlCmnSetSendVAP()` with the required data to be passed.

Eventually LBD issues an ioctl “`IEEE80211_IOCTL_DBGREQ`” with a command “`IEEE80211_DBGREQ_SENDBSTMREQ_TARGET`” to Driver with the structure populated with transition reason.

The details of added mapping function from `steerexec_reason` to MBO reason are as follows :

1. `state->reason` is of the enum type “`steerexec_reason_e`” which is internal to LBD.
2. This `state->reason` is mapped to a variable `steerMboReason` which is of enum type “`IEEE80211_BSTM_REQ_REASON_CODE`”, which is defined in Driver code.
3. The function returns this mapped `steerMboReason`.

MBO Reason Code 4 (Load Balancing) <--> { *steerexec\_reason\_activeOffload* }

MBO Reason Code 5 (Low RSSI) <--> { *steerexec\_reason\_activeDowngradeRSSI or steerexec\_reason\_idleDowngrade* }

MBO Reason Code 3 (Insufficient BW) <--> { *steerexec\_reason\_activeDowngradeRate* }

MBO Reason Code 9 (Premium AP) <--> { *steerexec\_reason\_activeUpgrade* }

MBO Reason Code 7 (High Interference) <--> { *steerexec\_reason\_interferenceAvoidance* }

MBO Reason Code 0 (Unspecified) <--> { rest of the reason in LBD }

### **Code changes in Wi-Fi for adding MBO reason code**

The ioctl “IEEE80211\_IOCTL\_DBGREQ” issued by LBD with the command “IEEE80211\_DBGREQ\_SENDBSTMREQ\_TARGET” is handled by a function “`ieee80211dbg_sendbstmreq_target()`”.

In this function “`ieee80211dbg_sendbstmreq_target()`”, a structure of same type “`ieee80211_bstm_reqinfo_target`” is defined and data (which now contains `trans_reason`) received from LBD is stored in this structure.

The aforementioned function makes function call to “`→ wlan_send_bstmreq_target() → ieee80211_send_bstm_req_target()`”.

Now, the function “`ieee80211_send_bstm_req_target()`” constructs the whole BSTM Request to be sent to STA. Two functions are defined to append MBO IE and MBO Attribute Transition Reason Code in the end of the prepared BTM request.

The functions are called “`ieee80211_setup_mbo_ie_bstmreq_target()`” and “`ieee80211_setup_trans_reason_ie_target()`”, which takes the structure sent by LBD and uses `trans_reason` field in it to add MBO reason Code in BTM request.

After these two functions add MBO IE and Transition Reason Code MBO Attribute to BTM Request, finally “`ieee80211_finalize_and_send_bstm_req()`” function is called with the BTM Request to be sent to STA.

### **Code changes in Wi-Fi for passing MBO reject reason code to lbd**

“`ieee80211_recv_bstm_resp()`” is the handler for the event “IEEE80211\_ACTION\_BSTM\_RESP” which processes the received BSTM Response from STA and passes the response to LBD.

The function “`ieee80211_recv_bstm_resp()`” checks whether Steer was successful or not and based on that it extracts the MAC Address of the target BSSID to which STA steered and then a case is added to extract MBO Transition Reject Reason Code and update in the node structure. A field is added in node structure namely `trans_reject_code`. This field is inside `ni->ni_mbo` structure. This newly added field is populated by the sent MBO Reject Reason Code.

The BTM response is sent to LBD in the form of a populated structure of type “`bs_wnm_bstm_resp`”. A new field `reject_code` is added to this structure and is populated from the node structure’s updated MBO rejected reason Code.

After adding all the required information to this structure, this structure is sent to LBD through “`ieee80211_bsteering_send_wnm_bstm_resp_event()`”.

### **Code changes in lbd for receiving MBO reject reason code**

“steerexecImplCmnCreate()” creates a listener to the event “wlanif\_event\_btm\_response” with the handler “steerexecImplCmnHandleBTMResponseEvent()”. This handler is called whenever LBD receives a BTM Response from Driver.

Inside this handler routine, resp->status is checked to determine whether it was successful or not, meaning, whether STA accepted or rejected the Steer Request.

If STA rejected the request, then before calling “steerexecImplCmnSteerEndBTMFailure()”, the steer\_type is set as Best Effort and checking if the request was made for activeOffload. And in case the request was made for activeOffload, then those STAs are marked as loadbalancingAttemptedButRejected stations.

For marking the STAs, a new field in the node STA structure is added called “loadbalancingAttemptedButRejected”. This field is added in the inNetworkInfo structure within stahandle structure.

For marking, unmarking and accessing this field, the following three new functions are defined, namely, “stadbEntryMarkActiveLoadBalancingRejected()”, “stadbEntryUnmarkActiveLoadBalancingRejected()” and “stadbEntryReadActiveLoadBalancingRejected”. These functions can be used from other modules in LBD to access and change the field value.

A new steer\_type called “steerexecImplCmnSteeringType\_btm\_be\_active\_force\_steer” is introduced, which is used in the later part of implementation.

### **Code changes in Wi-Fi driver for disassociation support**

“ieee80211\_bstm\_reqinfo\_target” structure has two new additional fields called “disassoc” and “disassoc\_timer” to be used by LBD to prepare BTM Request.

These fields are populated by LBD and sent as data to driver and driver has to include these in the BTM request actually sent to STA. This is done in function “ieee80211\_send\_bstm\_req\_target()”. This function calls a function called “ieee80211\_create\_bstm\_req()” that prepares the BTM Request Frame except the MBO IE and MBO attribute. Disassoc and disassoc\_timer is added as an input argument to the function “ieee80211\_create\_bstm\_req()” instead of hardcoding them.

### **Code changes in lbd for disassociation support**

“steerexecImplCmnDetermineBTMSteeringType()” decides the steerType of that is going to be performed. In this function if the STA is active and “stadbEntryReadActiveLoadBalancingRejected()” returns markedForceSteer, then the steerType is set to steerexecImplCmnSteeringType\_btm\_be\_active\_force\_steer.

Then in the function “steerexecImplCmnSteerBTM()” the aforementioned set steerType is used to set a flag called “forceSteer” to 1 or 0.

This flag is added to function “wlanif\_sendBTMRequest()” as an input argument and this function calls the final function “wlanifBSteerControlSendBTMRequest()” which prepares and sends the BTM request to Driver. So, this flag is added in this function too.

In function “wlanifBSteerControlSendBTMRequest()”, when it prepares the BTM request, it fills the structure “ieee80211\_bstm\_reqinfo\_target”. Two fields are added in this structure “disassoc” and “disassoc\_timer” to populate based on the forceSteer flag value.

If this flag is 1, then disassoc will be 1 and disassoc\_timer will be 1 second, otherwise they will be zero.

If BTM Response comes positive and steer is complete before disassoc timer expires then loadBalancingAttemptedButRejected flag is placed in state forceSteerComplete and disassoc timer is unregistered.

Otherwise, if BTM response does not come or is rejected then, at the expiry of the disassoc timer LBD will send disassociate to the STA in the same manner as Legacy Steer and set the state of loadBalancingAttemptedButRejected as forceSteerComplete.

At every bandmon update event on channel utilization, the STAs marked as forceSteerComplete will be iterated over to steer them back to original Band if the original band is not overloaded now.

### 13.23 Support offload-based radios to connect to Wi-Fi hotspot of phones

Offload (OL)-based radios fail to connect Wi-Fi hotspot of phones under STA modes as because certain phones respond to broadcast probe requests only and do not respond to unicast probe requests. Such problems occurred on certain phones on QCA9980 and IPQ40xx chipsets.

To resolve such problems, support is implemented for sending probe requests through broadcast method for OL-based radios under 3-address STA modes connecting to the phone hotspot. This capability is currently implemented for 2 GHz bands as such phones support only 2 GHz bands for hotspot feature, although this functionality can be extended to 5 GHz bands in the future. 3-address STA modes of OL radios must include QWRAP and ExtAP for this functionality.

When the STA VAP goes up, supplicant starts the scan and driver updates the new scan table entry from beacon and probe response. After the scan is completed, the following sequence of events occur:

- Supplicant initiates connection with matched BSSID from new scan table entry and sends unicast probe request and waiting for probe response from matched AP for approximately 600 ms.
- If probe response is not received for this unicast probe request, STA sends probe request with broadcast destination (DST) address and waits again for probe response. If the probe response is still not received for this broadcast probe request, connection failure notification is sent to supplicant and then supplicant initiates connection with another BSSID match if any from scan table entry. Otherwise, supplicant flushes the scan table and starts the new scan. The same steps are repeated.
- After the STA receives matched probe response for unicast probe request or broadcast probe request, STA sends success notification to supplicant. Supplicant starts authentication and performs association to complete the connection state.

Although this implementation is for offload, this design enhancement is radio/platform independent.

## 13.24 QCN broadcast probe response

The broadcast probe response capability is supported for Qualcomm (QCN)-based access points (APs) and stations.

### 13.24.1 Requirements addressed with QCN broadcast probe response

The following requirements are addressed using the QCN broadcast probe response feature:

- STA indicates that it is a Qualcomm STA using QCN IE. It also mentions the channel time IE if it sends a broadcast probe request.
- While receiving a probe request from a STA that indicates support for broadcast probe responses, collect probe requests from multiple STAs over a specified period of time and then respond to all of them together with one broadcast probe response.
- Respond with a broadcast probe response before channel time of all the STAs expires.
- If there is a beacon to be scheduled by AP before the channel time expires, send a beacon instead of broadcast probe response.
- While receiving a probe request from a STA that sends the directed probe request, send the unicast response immediately if the STA does not mention the channel time. If it mentions the channel time, drop the response if there is a beacon to be sent out before channel time expiry or send the unicast probe response.

### 13.24.2 Guidelines for QCN broadcast probe response

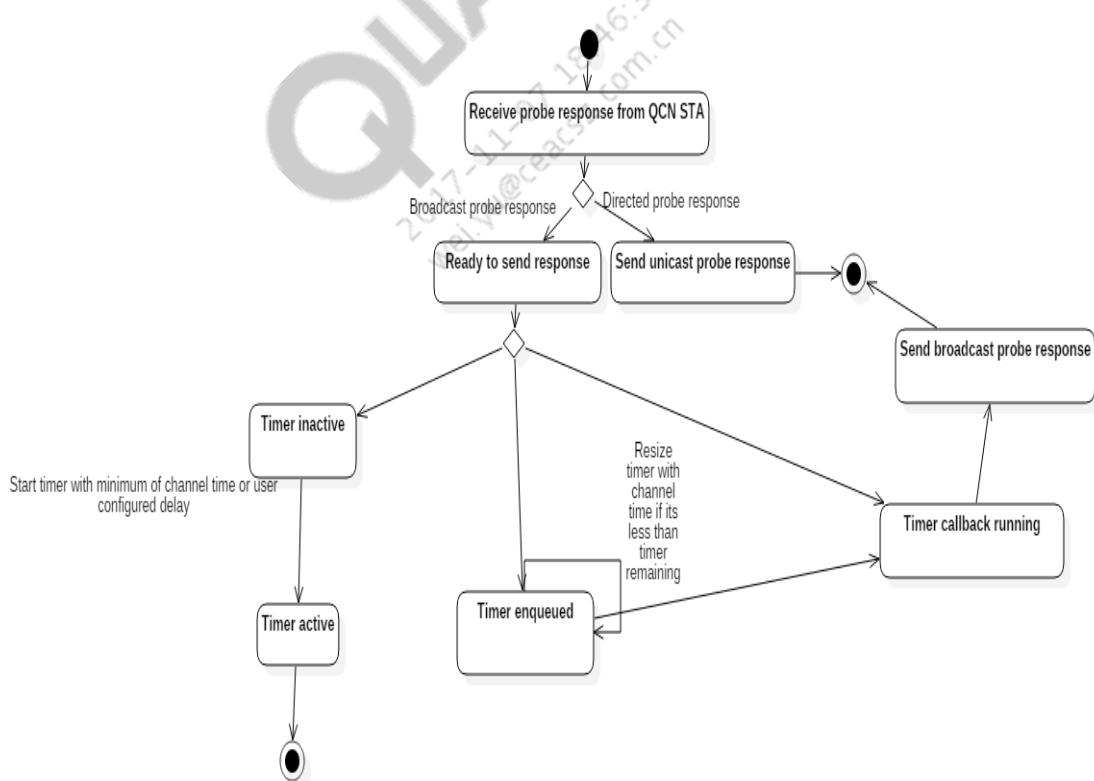
- The feature is enabled by default. However, a user can disable the feature during runtime using a driver IOCTL.
- Both STA and AP must be Qualcomm-based (support QCN\_IE) in the probe request.
- The broadcast response is always sent in the legacy rate.
- This feature is useful only when sufficient STAs that support it are present.
- The feature is supported only on platforms that support high resolution timers.
- This feature sends out the broadcast probe responses at legacy rates currently. This behavior is applicable to the QCN STAs only.

### 13.24.3 Design changes for QCN broadcast probe response

1. The STA sends the Qualcomm Information Element (QCN\_IE) in the directed or broadcast probe request. The STA may also send the channel time IE in the directed or broadcast probe request
2. In the receive probe request path, the IE is parsed and checked if it is a directed or broadcast probe request.
3. If unicast probe request and no channel time is specified, immediately unicast probe response is sent. If any channel time is specified, AP checks if there is a beacon to be scheduled before

the channel time of the STA expires. If there is any beacon, AP sends it and drops the unicast probe response.

4. If it is a broadcast probe request received from the STA, a timer is started with the user specified duration if channel time is not present. Else if channel time is present and lesser than the user specified value, then timer is started with channel time value. At the end of the timer, the callback is executed that sends out a broadcast probe response with the legacy rate supported by the vap.
5. Higher resolution timer is used here to achieve higher resolution in terms of microseconds because the channel time in STA is of the order of few milliseconds. The hrtimer always runs in hardirq context. The hrtimer callback, that runs in hardirq context, sends out a probe problems(i.e.) the timer callback to run in tasklet context and to be having higher resolution, **tasklet\_hrtimer** is used to achieve this. The timer here runs in tasklet context.
6. When the first broadcast probe request is received, start the timer with the minimum of user specified delay or the channel time of the STA. For the successive probe requests, resize the timer with the minimum of the time remaining or channel time. When the timer expires, send out the broadcast probe response. At any point in the state diagram mentioned below, the Software Beacon Alert (SWBA) handler will cancel the timer if timer callback is not running and a beacon has been scheduled before the timer expiration. Then, the timer will go to initial inactive state. The following diagram illustrates the state of the AP.



### 13.24.4 Configure QCN broadcast probe response capability

The following are the input/output controls (IOCTLS) introduced in the form of iwpriv commands to support the QCN broadcast probe response functionality:

| Parameter          | Command                        | Description   |
|--------------------|--------------------------------|---|
| bpr_disable 0/1    | iwpriv athX bpr_disable 0/1    | The default is 0 (enabled). 0 denotes enabling BCAST probe response feature and 1 denotes disabling BCAST probe response  |
| get_bpr_disable    | iwpriv athX get_bpr_disable    | Retrieves the bpr_disable value.  |
| bpr_delay [1-255]  | iwpriv athX bpr_delay [1-255]  | Denotes how long BCAST probe response is held by the AP. The default is 50ms. Takes values in ms. The range is between 1 and 255 ms.  |
| get_bpr_delay      | iwpriv athX get_bpr_delay      | Retrieves the delay value of broadcast probe responses.   |
| bpr_latency [1-10] | iwpriv athX bpr_latency [1-10] | Denotes before how much time (in ms) should the AP send the broadcast response before the channel time of STA expires. This parameter is configured per radio because it is the same across all VAPs per radio.<br>The default is 5ms; the range is between 1 and 10.   |
| get_bpr_latency    | iwpriv athX get_bpr_latency    | Retrieves the latency value of broadcast probe responses.   |
| bcn_latency [5-20] | iwpriv athX bcn_latency [5-20] | Denotes the time (in ms) to compensate for the delay caused due to beacon sent from the driver to firmware. This parameter is configured per radio because it is the same across all VAPs per radio.<br>The default is 5ms; the range is between 5 and 20.  |
| get_bcn_latency    | iwpriv athX get_bcn_latency    | Retrieves the latency value of beacons.   |
| clr_bpr_stats      | iwpriv athX clr_bpr_stats      | Clears all the timer-related counters.<br>The sample output is as follows:<br>BPR timer start count - 0<br>BPR timer resize count - 0<br>BPR timer callback count - 0<br>BPR timer cancel count - 0   |
| get_bpr_stats      | iwpriv athX get_bpr_stats      | Prints the summary of all the data.<br>The sample output is as follows:<br>BPR feature disabled - 0<br>BPR Latency compensation - 5 ms<br>Beacon Latency compensation- 5 ms<br>BPR delay - 50 ms<br>Current Timestamp -<br>16179117804480<br>Next beacon Timestamp -<br>16179126788320<br>BPR timer start count - 20<br>BPR timer resize count - 2<br>BPR timer callback count - 10<br>BPR timer cancel count - 7 |

### 13.24.5 Sample configuration scenario

Consider a scenario in which an AP with a VAP is brought up. If a user connects any legacy STA, which causes the STA to receive a unicast probe response immediately. If the user connects a QCN STA, a unicast probe request from it must be served immediately because there is no channel time.

Alternatively, if a user connects a QCN STA and a broadcast probe request is sent from it, a timer is started. If a beacon is present before the timer expires, either a beacon is sent by AP or the AP sends a broadcast probe response.

As simulating multiple QCN STA scenarios are difficult in this case (of the order of milliseconds), the channel time is preset and hard-coded for all the incoming broadcast requests to proceed with the testing and to determine whether the behavior complies with the aforementioned state diagram in this section.

## 13.25 Configure Tx Ack timeout value

A mechanism to specify the Tx acknowledgment (Ack) packet timeout value is available. Enter the `iwpriv wifiX acktimeout yy` command to specify the Tx acknowledgment packet timeout value. This command is based on the `OL_ATH_PARAM_TXACKTIMEOUT` to create. The `OL_ATH_PARAM_TXACKTIMEOUT` parameter is added for physical device (PDEV). Here, `wifiX` is wifi interface `wifi0/wifi1/wifi2`, and `yy` is the Tx ACK timeout value where `yy` ranges from `0x40–0xff`. If the value less than `0x40` or big than `0xff`, it prints out the assert command. This value will save to `tx_ack_timeout` parameter.

This `iwpriv wifiX acktimeout yy` command calls the `WMI_PDEV_PARAM_TX_ACK_TIMEOUT` WMI command to send the `yy` value to FW.

This capability is supported on the QCA9984, QCA9980, QCA98886, and IPQ4019 platforms; it is not supported on the QCA9880 chipsets. When AP is rebooted or powered on again, reset this value again manually. The Tx Ack value ranges from `0x40` to `0xFF`. When FW recovers, it is reset to `0x40` and needs to be modified accordingly manually. The Tx packet retry latency increases as timeout value changes to a value higher than `0x40` and causes performance degree in some environments.

Enter the `iwpriv wifiX get_acktimeout` command to retrieve the configured Tx Ack timeout value.

### 13.25.1 Configure and verify the Tx Ack timeout value

To configure and verify the Tx Ack timeout value, do the following:

1. Enable the Wi-Fi interface using `wifi up` option with the following UCI commands:

```
uci set wireless.wifi0.disabled=0
uci set wireless.wifi1.disabled=0
uci commit wireless
wifi up
```

The `ath0` and `ath1` interfaces must be up with SSID OpenWrt for 2.4 GHz and 5 GHz for default value.

2. Use the following commands to check the default TX\_ACK\_TIMEOUT value.

```
athdiag -wifi=0 -get -address=0x36000
athdiag -wifi=1 -get -address=0x36000
```

Bit0 to Bit7 must display 40 for both interfaces.

3. Use the following commands to set the new value for TX ACK timeout.

```
iwpriv wifi0 acktimeout 0x50
iwpriv wifi1 acktimeout 0x60
```

The commands must not show any error. athdiag -wifi=0 -get -address=0x36000 to check if bit0 to bit7 is 50. If not, test is failed. athdiag -wifi=1 -get -address=0x36000 to check if bit0 to bit7 is 60. If not, test is failed. Use iwpriv wifi0 get\_acktimeout to check if the value shows 0x50. If not, the test is failed. Use iwpriv wifi1 get\_acktimeout to check if the value shows 0x50. If not, the test is failed.

4. Use the following commands to set the new value for TX ACK timeout:

```
iwpriv wifi0 acktimeout 0x100
iwpriv wifi1 acktimeout 0x20
```

The console must display appropriate error messages to indicate the value to be set between 0x40 and 0xFF.

### 13.25.2 Verify that ACK timeout value is not modified after channel change

Use the `iwpriv wifiX acktimeout yy` command to reset the ACK timeout value and after a change of channel, the Tx Ack timeout value must not be modified.

1. Enable the Wi-Fi interface using the `wifi up` option with the UCI commands:

```
uci set wireless.wifi0.disabled=0
uci set wireless.wifi1.disabled=0
uci commit wireless
wifi up
```

`ath0` and `ath1` interface must be up with ssid OpenWrt for 2.4 GHz and 5 GHz for default values.

2. Use the following commands to check the default TX\_ACK\_TIMEOUT value.

```
athdiag -wifi=0 -get -address=0x36000
athdiag -wifi=1 -get -address=0x36000
```

Bit0 to Bit7 must show 40 for both interfaces.

3. Use the following commands to set the new value for Tx ACK timeout.

```
iwpriv wifi0 acktimeout 0x50
iwpriv wifi1 acktimeout 0x60
```

`athdiag -wifi=0 -get -address=0x36000` to check if bit0 to bit7 is 50. If not, test is failed.

`athdiag -wifi=1 -get -address=0x36000` to check if bit0 to bit7 is 60. If not, test is failed.

Use `iwpriv wifi0 get_acktimeout` to check if the value shows 0x50. If not, the test is failed. Use `iwpriv wifi1 get_acktimeout` to check if the value shows 0x50. If not, the test is failed.

4. Use the following commands to change the channel:

```
ifconfig ath0 down
ifconfig ath1 down
iwconfig ath0 channel x
iwconfig ath1 channel x
ifconfig ath0 up
ifconfig ath1 up
```

athdiag -wifi=0 -get -address=0x36000 to check if bit0 to bit7 is 50. If not, test is failed.  
 athdiag -wifi=1 -get -address=0x36000 to check if bit0 to bit7 is 60. If not, test is failed.

Use iwpriv wifi0 get\_acktimeout to check if the value shows 0x50. If not, the test is failed. Use iwpriv wifil get\_acktimeout to check if the value shows 0x50. If not, the test is failed.

## 13.26 UCI commands to configure RADIUS parameters for retries

The following RADIUS parameters to configure the number of times for which an attempt is made to establish a connection with the RADIUS server, and the frequency or the interval at which these retries are made can be set using UCI commands:

- identity\_request\_retry\_interval ( Range 0 to 20 )
- radius\_server\_retries ( Range 0 to 10 )

Enter the following command to specify a RADIUS server retry interval of 3 seconds:

```
uci set wireless.@wifi-iface[0].identity_request_retry_interval =3
```

Enter the following command to specify the number of retries to be attempted with the RADIUS server as 5:

```
uci set wireless.@wifi-iface[0].radius_server_retries =5
```

Commit the settings:

```
uci commit wireless
```

The following is an example of a UCI configuration file with the RADIUS server retry settings:

```
root@OpenWrt:/# vi /etc/config/wireless

config wifi-device 'wifi0'
    option type 'qcawifi'
    option macaddr '8c:fd:f0:01:42:c4'
    option hwmode '11ac'
    option disabled '0'
    option htmode 'HT80'
    option channel '36'
    option txchainmask '7'
    option rxchainmask '7'

config wifi-device 'wifil'
    option type 'qcawifi'
```

```

        option channel 'auto'
        option macaddr '8c:fd:f0:00:bd:f3'
        option hwmode '11ng'
        option disabled '1'

    config wifi-iface
        option network 'lan'
        option mode 'ap'
        option ssid 'SecurityMat_3'
        option encryption 'wpa2+ccmp'
        option device 'wifi0'
        option server '192.168.1.150'
        option port '1812'
        option key 'atheros123'
        option nss '3'
        option identity_request_retry_interval '3'
        option radius_server_retries '5'

```

Configure the AP in WPA2 EAP-TLS – AES CCMP using QSPR. Configure the identity\_request\_retry\_interval and radius\_server\_retries param values using UCI. Bring down the Ethernet interface so that the station cannot authenticate with the RADIUS server. Start the hostapd with debug enabled. Check for the maximum retry attempts in hostapd log . The default maximum retry attempts without the configuration is 10.

#### **RADIUS: Retry attempts :2 Maximum retry attempts :5**

ath0: RADIUS Next RADIUS client retransmit in 1 seconds

ath0: STA 00:03:7f:40:00:2e RADIUS: Resending RADIUS message (id=0)

#### **RADIUS: Retry attempts :4 Maximum retry attempts :5**

ath0: RADIUS Next RADIUS client retransmit in 5 seconds

ath0: STA 00:03:7f:40:00:2e RADIUS: Resending RADIUS message (id=1)

Check for identity\_request\_retry\_interval . In default cases, time is calculated based on dynamic backoff and keeps increasing. The identity\_request\_retry\_interval value is saved to the value specified using the parameter in UCI.

## 13.27 WAPI with 16 VAP clients

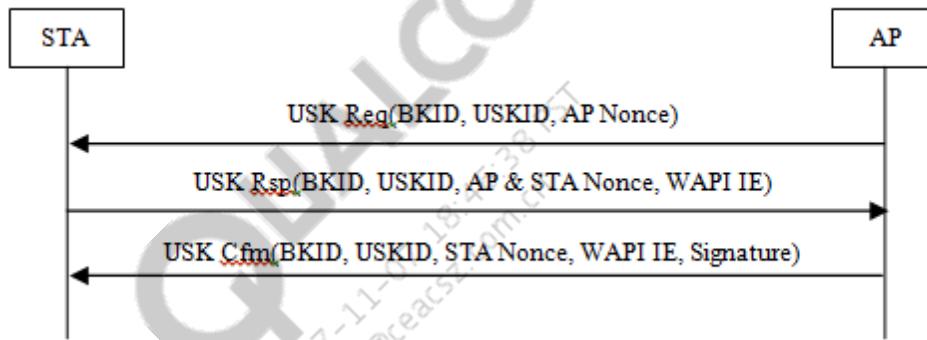
WLAN Authentication and Privacy Infrastructure (WAPI) is a Chinese National Standard for wireless LANs. It can be regarded as another Wi-Fi security protocol in this document. WAPI encryption is supported in QCA9984 hardware. The cipher engine is located in the hardware, but firmware need to configure the hardware registers so that the hardware knows when and how to apply WAPI encryption method.

In QCA9984, WAPI should be supported regardless WDS is enabled or not. When it is an AP, a station should support up to 16 VAPs with WAPI enabled for these VAPs. A STA should be able to connect anyone of these 16 VAPs.

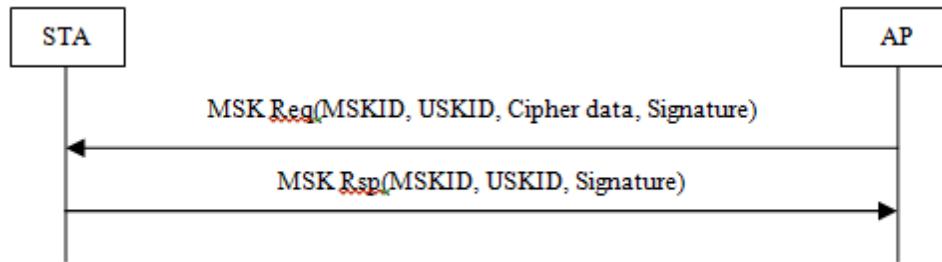
Interoperability among **WAPI** (WLAN authentication and Privacy Infrastructure) products that support HT PHY is available. It includes various combinations of MAC and HT physical features. The MAC features include beacon interval, ESSID, RTS/CTS, fragment and WAPI authentication and encryption on a selected channel. The HT physical features include AMPDU, AMSDU, short GI, 20/40MHz channel bandwidth, and different tx/rx chain.

APs and STAs must support the receipt of the fragmented packets.

For WAPI-PSK authentication and encryption management, PSK must support 8 – 63 octet ascii or hexadecimal characters pass-phase APs and STAs must support the **USK** (unicast session key) three-way handshakes.



APs must support to generate the **MSK** (multicast session key), and optional to support update MSK.

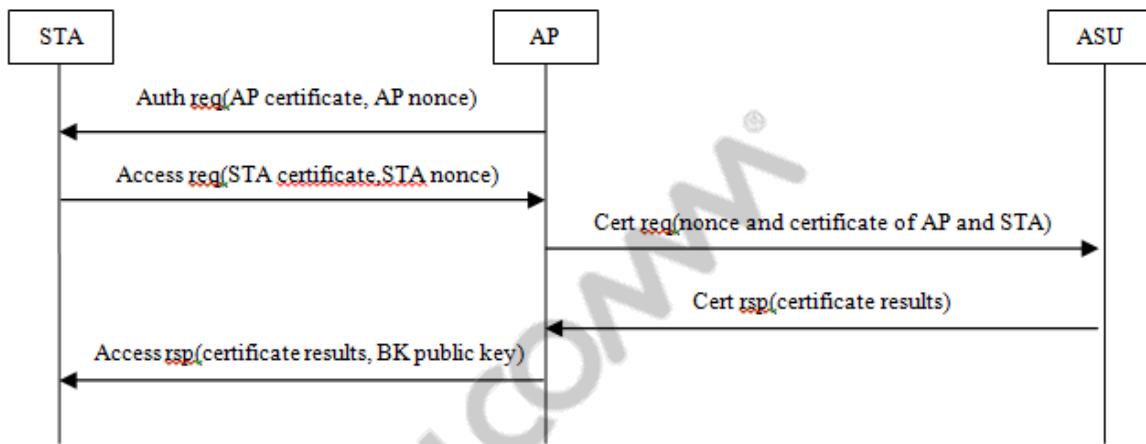


For WAPI certificate authentication and encryption management, enterprise APs must support **X509v3** (mandatory, PKI public key infrastructure and PMI privilege management infrastructure, file of .pem, .cer. The signature algorithm is 192-bit ECDSA. Hash algorithm is SHA256) certificate, and optional to support **GBW** (optional,) certificate.

Enterprise APs must support to use other ASU to authenticate the user and server certificate

Enterprise STAs must support X509v3 certificate, and optional to support GBW certificate

Enterprise APs and Enterprise STAs must support to generate BK by certificate authentication process.



**Figure 13-26 Three-certificate mode**

For one Tx chain and one Rx chain ( $1 \times 1$ ),  $1 \times 1$  APs must set the “Supported MCS Set field” to MCS 0 - 7 in HT Capabilities element of beacon frame.  $1 \times 1$  STAs must set the “Supported MCS Set field” to MCS 0 - 7 in HT Capabilities element of association request frame.  $2 \times 2$  STA must act as a  $1 \times 1$  STA when associating with an  $1 \times 1$  AP

For 2 Tx chain and 2 Rx chain ( $2 \times 2$ ),  $2 \times 2$  APs must set the “Supported MCS Set field” to MCS 0 – 15 in HT Capabilities element of beacon frame.  $2 \times 2$  STAs must set the “Supported MCS Set field” to MCS 0 – 15 in HT Capabilities element of association request frame.

In 40MHz channel bandwidth BSS, AP must set the “Supported channel width set” to 1 in HT Capabilities element of beacon frame, also set the “STA Channel Width field” and “Channel offset field” to use SCB or SCA as the secondary channel in HT Operation element of beacon frame.

In 20MHz channel bandwidth BSS, AP must set the “Supported channel width set” to 0 in HT Capabilities element of beacon frame, also set the “STA Channel Width field” to 0 in HT Operation element of beacon frame.

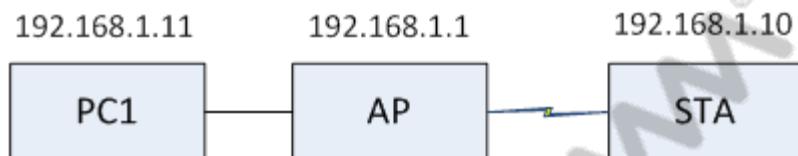
An HT STA/AP indicate the maximum AMPDU in the Maximum AMPDU Length Exponent field in HT Capabilities element. An HT STA shall not transmit an AMPDU containing an MPDU with a group addressed RA. An HT AP shall not transmit an AMPDU containing group addressed MPDUs if the HT Protection field is set to non-HT mixed mode. AMPDU must response with a HT immediate or delayed block ack frame.

GI changes to 400ns

WDS is not supported in this configuration scenario of WAPI with VAP clients. In QCA9980, only one vap can be configured in WAPI security.

### 13.27.1 Sample configuration scenarios

Consider the following configuration deployment:



#### 13.27.1.1 One VAP, one STA, WDS Disabled

This sample scenario is to verify whether WAPI works when WDS is disabled with only one VAP on AP side. The following is the AP configuration:

```

uci set wireless.wifi2=wifi-device
uci set wireless.wifi2.type=qcawifi
uci set wireless.wifi2.macaddr=8C:FD:02:00:c9:c9
uci set wireless.wifi2.hwmode=11ac
uci set wireless.wifi2.disabled=0
uci set wireless.wifi2.htmode=HT80
uci set wireless.wifi2.channel=36
uci set wireless.wifi2.txchainmask=15
uci set wireless.wifi2.rxchainmask=15
uci set wireless.wifi2.mode=ap
uci set wireless.@wifi-iface[0]=wifi-iface
uci set wireless.@wifi-iface[0].device=wifi2
uci set wireless.@wifi-iface[0].network=lan
uci set wireless.@wifi-iface[0].mode=ap
uci set wireless.@wifi-iface[0].ssid=5g_wapi
uci set wireless.@wifi-iface[0].encryption=wapi-psk
uci set wireless.@wifi-iface[0].key=12345600
uci commit wireless
    
```

WAPI client need to be used to perform the test. After AP and STA associated, do following on STA: *Assign IP Address 192.168.1.10 to wapi client*. Execute from AP: *ping 192.168.1.10*. Execute from STA: *ping 192.168.1.11*.

AP and STA associated and able to ping each other.

### 13.27.1.2 16 VAP with WDS Disable

This sample scenario is to verify whether WAPI works when there are 16 VAPs on AP with WDS disable. The following is the AP configuration:

```
uci set wireless.wifi2.bcnburst=1
uci set wireless.wifi2=wifi-device
uci set wireless.wifi2.type=qcawifi
uci set wireless.wifi2.macaddr=8C:FD:02:00:c9:c9
uci set wireless.wifi2.hwmode=11ac
uci set wireless.wifi2.disabled=0
uci set wireless.wifi2.htmode=HT80
uci set wireless.wifi2.channel=100
uci set wireless.wifi2.txchainmask=15
uci set wireless.wifi2.rxchainmask=15
uci set wireless.wifi2.mode=ap
uci set wireless.@wifi-iface[0]=wifi-iface
uci set wireless.@wifi-iface[0].device=wifi2
uci set wireless.@wifi-iface[0].network=lan
uci set wireless.@wifi-iface[0].mode=ap
uci set wireless.@wifi-iface[0].wds=0
uci set wireless.@wifi-iface[0].ssid=5g_wapi
uci set wireless.@wifi-iface[0].encryption=wapi-psk
uci set wireless.@wifi-iface[0].key=12345600
uci add wireless wifi-iface
uci set wireless.@wifi-iface[1].device=wifi2
uci set wireless.@wifi-iface[1].network=lan
uci set wireless.@wifi-iface[1].mode=ap
uci set wireless.@wifi-iface[1].wds=0
uci set wireless.@wifi-iface[1].ssid=5g_wapi_1
uci set wireless.@wifi-iface[1].encryption=wapi-psk
uci set wireless.@wifi-iface[1].key=12345601
uci add wireless wifi-iface
uci set wireless.@wifi-iface[2].device=wifi2
uci set wireless.@wifi-iface[2].network=lan
uci set wireless.@wifi-iface[2].mode=ap
uci set wireless.@wifi-iface[2].wds=0
uci set wireless.@wifi-iface[2].ssid=5g_wapi_2
uci set wireless.@wifi-iface[2].encryption=wapi-psk
uci set wireless.@wifi-iface[2].key=12345602
uci add wireless wifi-iface
uci set wireless.@wifi-iface[3].device=wifi2
uci set wireless.@wifi-iface[3].network=lan
uci set wireless.@wifi-iface[3].mode=ap
```

```
uci set wireless.@wifi-iface[3].wds=0
uci set wireless.@wifi-iface[3].ssid=5g_wapi_3
uci set wireless.@wifi-iface[3].encryption=wapi-psk
uci set wireless.@wifi-iface[3].key=12345603
uci add wireless wifi-iface
uci set wireless.@wifi-iface[4].device=wifi0
uci set wireless.@wifi-iface[4].network=lan
uci set wireless.@wifi-iface[4].mode=ap
uci set wireless.@wifi-iface[4].wds=0
uci set wireless.@wifi-iface[4].ssid=5g_wapi_4
uci set wireless.@wifi-iface[4].encryption=wapi-psk
uci set wireless.@wifi-iface[4].key=12345604
uci add wireless wifi-iface
uci set wireless.@wifi-iface[5].device=wifi2
uci set wireless.@wifi-iface[5].network=lan
uci set wireless.@wifi-iface[5].mode=ap
uci set wireless.@wifi-iface[5].wds=0
uci set wireless.@wifi-iface[5].ssid=5g_wapi_5
uci set wireless.@wifi-iface[5].encryption=wapi-psk
uci set wireless.@wifi-iface[5].key=12345605
uci add wireless wifi-iface
uci set wireless.@wifi-iface[6].device=wifi2
uci set wireless.@wifi-iface[6].network=lan
uci set wireless.@wifi-iface[6].mode=ap
uci set wireless.@wifi-iface[6].wds=0
uci set wireless.@wifi-iface[6].ssid=5g_wapi_6
uci set wireless.@wifi-iface[6].encryption=wapi-psk
uci set wireless.@wifi-iface[6].key=12345606
uci add wireless wifi-iface
uci set wireless.@wifi-iface[7].device=wifi2
uci set wireless.@wifi-iface[7].network=lan
uci set wireless.@wifi-iface[7].mode=ap
uci set wireless.@wifi-iface[7].wds=0
uci set wireless.@wifi-iface[7].ssid=5g_wapi_7
uci set wireless.@wifi-iface[7].encryption=wapi-psk
uci set wireless.@wifi-iface[7].key=12345607
uci add wireless wifi-iface
uci set wireless.@wifi-iface[8].device=wifi2
uci set wireless.@wifi-iface[8].network=lan
uci set wireless.@wifi-iface[8].mode=ap
uci set wireless.@wifi-iface[8].wds=0
uci set wireless.@wifi-iface[8].ssid=5g_wapi_8
```

```
uci set wireless.@wifi-iface[8].encryption=wapi-psk
uci set wireless.@wifi-iface[8].key=12345608
uci add wireless wifi-iface
uci set wireless.@wifi-iface[9].device=wifi2
uci set wireless.@wifi-iface[9].network=lan
uci set wireless.@wifi-iface[9].mode=ap
uci set wireless.@wifi-iface[9].wds=0
uci set wireless.@wifi-iface[9].ssid=5g_wapi_9
uci set wireless.@wifi-iface[9].encryption=wapi-psk
uci set wireless.@wifi-iface[9].key=12345609
uci add wireless wifi-iface
uci set wireless.@wifi-iface[10].device=wifi2
uci set wireless.@wifi-iface[10].network=lan
uci set wireless.@wifi-iface[10].mode=ap
uci set wireless.@wifi-iface[10].wds=0
uci set wireless.@wifi-iface[10].ssid=5g_wapi_10
uci set wireless.@wifi-iface[10].encryption=wapi-psk
uci set wireless.@wifi-iface[10].key=12345610
uci add wireless wifi-iface
uci set wireless.@wifi-iface[11].device=wifi2
uci set wireless.@wifi-iface[11].network=lan
uci set wireless.@wifi-iface[11].mode=ap
uci set wireless.@wifi-iface[11].wds=0
uci set wireless.@wifi-iface[11].ssid=5g_wapi_11
uci set wireless.@wifi-iface[11].encryption=wapi-psk
uci set wireless.@wifi-iface[11].key=12345611
uci add wireless wifi-iface
uci set wireless.@wifi-iface[12].device=wifi2
uci set wireless.@wifi-iface[12].network=lan
uci set wireless.@wifi-iface[12].mode=ap
uci set wireless.@wifi-iface[12].wds=0
uci set wireless.@wifi-iface[12].ssid=5g_wapi_12
uci set wireless.@wifi-iface[12].encryption=wapi-psk
uci set wireless.@wifi-iface[12].key=12345612
uci add wireless wifi-iface
uci set wireless.@wifi-iface[13].device=wifi2
uci set wireless.@wifi-iface[13].network=lan
uci set wireless.@wifi-iface[13].mode=ap
uci set wireless.@wifi-iface[13].wds=0
uci set wireless.@wifi-iface[13].ssid=5g_wapi_13
uci set wireless.@wifi-iface[13].encryption=wapi-psk
uci set wireless.@wifi-iface[13].key=12345613
```

```

uci add wireless wifi-iface
uci set wireless.@wifi-iface[14].device=wifi2
uci set wireless.@wifi-iface[14].network=lan
uci set wireless.@wifi-iface[14].mode=ap
uci set wireless.@wifi-iface[14].wds=0
uci set wireless.@wifi-iface[14].ssid=5g_wapi_14
uci set wireless.@wifi-iface[14].encryption=wapi-psk
uci set wireless.@wifi-iface[14].key=12345614
uci add wireless wifi-iface
uci set wireless.@wifi-iface[15].device=wifi2
uci set wireless.@wifi-iface[15].network=lan
uci set wireless.@wifi-iface[15].mode=ap
uci set wireless.@wifi-iface[15].wds=0
uci set wireless.@wifi-iface[15].ssid=5g_wapi_15
uci set wireless.@wifi-iface[15].encryption=wapi-psk
uci set wireless.@wifi-iface[15].key=12345615
uci commit wireless

```

WAPI client need to be used to perform the test. After AP and STA associated, do following on STA: *Assign IP Address 192.168.1.10 to the wapi client.* Execute from AP: *ping 192.168.1.10.* Execute from STA: *ping 192.168.1.11.*

Connect ssid=5g\_wapi\_1 and provide the key=12345601. Connect ssid=5g\_wapi\_5 and provide the key=12345605. Connect ssid=5g\_wapi\_10 and provide the key=12345610. Connect ssid=5g\_wapi\_11 and provide the key=12345611. Connect ssid=5g\_wapi\_15 and provide the key=12345615

AP and STA must be able to ping each other.

### 13.27.1.3 16 VAP with different Security mode

This sample scenario is to verify whether WAPI works for different security mode (WPA+PSK2 and WEP). The following is the AP configuration:

```

uci set wireless.wifi0.bcnburst=1
uci set wireless.wifi0=wifi-device
uci set wireless.wifi0.type=qcawifi
uci set wireless.wifi0.macaddr=8C:FD:02:00:c9:c9
uci set wireless.wifi0.hwmode=11ac
uci set wireless.wifi0.disabled=0
uci set wireless.wifi0.htmode=HT80
uci set wireless.wifi0.channel=100
uci set wireless.wifi0.txchainmask=15
uci set wireless.wifi0.rxchainmask=15
uci set wireless.wifi0.mode=ap
uci set wireless.@wifi-iface[0]=wifi-iface

```

```
uci set wireless.@wifi-iface[0].device=wifi2
uci set wireless.@wifi-iface[0].network=lan
uci set wireless.@wifi-iface[0].mode=ap
uci set wireless.@wifi-iface[0].wds=0
uci set wireless.@wifi-iface[0].ssid=5g_wapi
uci set wireless.@wifi-iface[0].encryption=wpa-psk2
uci set wireless.@wifi-iface[0].key=12345600
uci add wireless wifi-iface
uci set wireless.@wifi-iface[1].device=wifi2
uci set wireless.@wifi-iface[1].network=lan
uci set wireless.@wifi-iface[1].mode=ap
uci set wireless.@wifi-iface[1].wds=0
uci set wireless.@wifi-iface[1].ssid=5g_wapi_1
uci set wireless.@wifi-iface[1].encryption=wep
uci set wireless.@wifi-iface[1].key=12345601
uci add wireless wifi-iface
uci set wireless.@wifi-iface[2].device=wifi2
uci set wireless.@wifi-iface[2].network=lan
uci set wireless.@wifi-iface[2].mode=ap
uci set wireless.@wifi-iface[2].wds=0
uci set wireless.@wifi-iface[2].ssid=5g_wapi_2
uci set wireless.@wifi-iface[2].encryption=wapi-psk
uci set wireless.@wifi-iface[2].key=12345602
uci add wireless wifi-iface
uci set wireless.@wifi-iface[3].device=wifi2
uci set wireless.@wifi-iface[3].network=lan
uci set wireless.@wifi-iface[3].mode=ap
uci set wireless.@wifi-iface[3].wds=0
uci set wireless.@wifi-iface[3].ssid=5g_wapi_3
uci set wireless.@wifi-iface[3].encryption=wapi-psk
uci set wireless.@wifi-iface[3].key=12345603
uci add wireless wifi-iface
uci set wireless.@wifi-iface[4].device=wifi2
uci set wireless.@wifi-iface[4].network=lan
uci set wireless.@wifi-iface[4].mode=ap
uci set wireless.@wifi-iface[4].wds=0
uci set wireless.@wifi-iface[4].ssid=5g_wapi_4
uci set wireless.@wifi-iface[4].encryption=wapi-psk
uci set wireless.@wifi-iface[4].key=12345604
uci add wireless wifi-iface
uci set wireless.@wifi-iface[5].device=wifi2
uci set wireless.@wifi-iface[5].network=lan
```

```
uci set wireless.@wifi-iface[5].mode=ap
uci set wireless.@wifi-iface[5].wds=0
uci set wireless.@wifi-iface[5].ssid=5g_wapi_5
uci set wireless.@wifi-iface[5].encryption=wapi-psk
uci set wireless.@wifi-iface[5].key=12345605
uci add wireless wifi-iface
uci set wireless.@wifi-iface[6].device=wifi2
uci set wireless.@wifi-iface[6].network=lan
uci set wireless.@wifi-iface[6].mode=ap
uci set wireless.@wifi-iface[6].wds=0
uci set wireless.@wifi-iface[6].ssid=5g_wapi_6
uci set wireless.@wifi-iface[6].encryption=wapi-psk
uci set wireless.@wifi-iface[6].key=12345606
uci add wireless wifi-iface
uci set wireless.@wifi-iface[7].device=wifi2
uci set wireless.@wifi-iface[7].network=lan
uci set wireless.@wifi-iface[7].mode=ap
uci set wireless.@wifi-iface[7].wds=0
uci set wireless.@wifi-iface[7].ssid=5g_wapi_7
uci set wireless.@wifi-iface[7].encryption=wapi-psk
uci set wireless.@wifi-iface[7].key=12345607
uci add wireless wifi-iface
uci set wireless.@wifi-iface[8].device=wifi2
uci set wireless.@wifi-iface[8].network=lan
uci set wireless.@wifi-iface[8].mode=ap
uci set wireless.@wifi-iface[8].wds=0
uci set wireless.@wifi-iface[8].ssid=5g_wapi_8
uci set wireless.@wifi-iface[8].encryption=wapi-psk
uci set wireless.@wifi-iface[8].key=12345608
uci add wireless wifi-iface
uci set wireless.@wifi-iface[9].device=wifi2
uci set wireless.@wifi-iface[9].network=lan
uci set wireless.@wifi-iface[9].mode=ap
uci set wireless.@wifi-iface[9].wds=0
uci set wireless.@wifi-iface[9].ssid=5g_wapi_9
uci set wireless.@wifi-iface[9].encryption=wapi-psk
uci set wireless.@wifi-iface[9].key=12345609
uci add wireless wifi-iface
uci set wireless.@wifi-iface[10].device=wifi2
uci set wireless.@wifi-iface[10].network=lan
uci set wireless.@wifi-iface[10].mode=ap
uci set wireless.@wifi-iface[10].wds=0
```

```
uci set wireless.@wifi-iface[10].ssid=5g_wapi_10
uci set wireless.@wifi-iface[10].encryption=wapi-psk
uci set wireless.@wifi-iface[10].key=12345610
uci add wireless wifi-iface
uci set wireless.@wifi-iface[11].device=wifi2
uci set wireless.@wifi-iface[11].network=lan
uci set wireless.@wifi-iface[11].mode=ap
uci set wireless.@wifi-iface[11].wds=0
uci set wireless.@wifi-iface[11].ssid=5g_wapi_11
uci set wireless.@wifi-iface[11].encryption=wapi-psk
uci set wireless.@wifi-iface[11].key=12345611
uci add wireless wifi-iface
uci set wireless.@wifi-iface[12].device=wifi2
uci set wireless.@wifi-iface[12].network=lan
uci set wireless.@wifi-iface[12].mode=ap
uci set wireless.@wifi-iface[12].wds=0
uci set wireless.@wifi-iface[12].ssid=5g_wapi_12
uci set wireless.@wifi-iface[12].encryption=wapi-psk
uci set wireless.@wifi-iface[12].key=12345612
uci add wireless wifi-iface
uci set wireless.@wifi-iface[13].device=wifi2
uci set wireless.@wifi-iface[13].network=lan
uci set wireless.@wifi-iface[13].mode=ap
uci set wireless.@wifi-iface[13].wds=0
uci set wireless.@wifi-iface[13].ssid=5g_wapi_13
uci set wireless.@wifi-iface[13].encryption=wapi-psk
uci set wireless.@wifi-iface[13].key=12345613
uci add wireless wifi-iface
uci set wireless.@wifi-iface[14].device=wifi2
uci set wireless.@wifi-iface[14].network=lan
uci set wireless.@wifi-iface[14].mode=ap
uci set wireless.@wifi-iface[14].wds=0
uci set wireless.@wifi-iface[14].ssid=5g_wapi_14
uci set wireless.@wifi-iface[14].encryption=wapi-psk
uci set wireless.@wifi-iface[14].key=12345614
uci add wireless wifi-iface
uci set wireless.@wifi-iface[15].device=wifi2
uci set wireless.@wifi-iface[15].network=lan
uci set wireless.@wifi-iface[15].mode=ap
uci set wireless.@wifi-iface[15].wds=0
uci set wireless.@wifi-iface[15].ssid=5g_wapi_15
uci set wireless.@wifi-iface[15].encryption=wapi-psk
```

```
uci set wireless.@wifi-iface[15].key=12345615
uci commit wireless
```

Use WAPI Client to connect with the AP VAP which is having security mode as “wapi-psk”, for other security (wpa-psk) mode QCA9980 board can be used as a client.

```
uci set wireless.wifi0=wifi-device
uci set wireless.wifi0.type=qcawifi
uci set wireless.wifi0.macaddr=84:60:aa:00:c9:cc
uci set wireless.wifi0.hwmode=11ac
uci set wireless.wifi0.disabled=0
uci set wireless.wifi0.htmode=HT80
uci set wireless.wifi0.channel=36
uci set wireless.wifi0.txchainmask=15
uci set wireless.wifi0.rxchainmask=15
uci set wireless.wifi0.mode=sta
uci set wireless.@wifi-iface[0]=wifi-iface
uci set wireless.@wifi-iface[0].device=wifi0
uci set wireless.@wifi-iface[0].network=lan
uci set wireless.@wifi-iface[0].mode=sta
uci set wireless.@wifi-iface[0].wds=0
uci set wireless.@wifi-iface[0].ssid=5g_wapi
uci set wireless.@wifi-iface[0].encryption=wpa-psk2
uci set wireless.@wifi-iface[0].key=12345600
uci commit wireless
```

After AP and STA associated, assign IP Address 192.168.1.10 to WAPI client. Execute from AP: ping 192.168.1.10. Execute from STA: ping 192.168.1.11. AP and STA should be able to ping each other.

Change following parameters on STA setup: ssid=5g\_wapi, key=12345600 and encryption=wpa-psk2. AP and STA associated, and are able to ping each other.

Change following parameters on STA setup: ssid=5g\_wapi\_1, key=12345601 and encryption=wep. AP and STA are associated, but AP is unable to ping the STA when QCA9980 board has been used as STA. However, AP becomes connected with mobile station and they can ping each other.

## 13.28 Configure random number generation mode

This section describes details on host design changes for improving random number generation. This enhancement is supported on all direct-attach chipsets.

Modifications have been made to IC structure to add mode variable (random\_gen\_mode) to store current mode. A iwpriv command is provided to change the modes instantaneously.

A kthread is started at the time of attach. This is responsible for updating /dev/random by rssi by ADC register values based on the mode. User can change the mode using the iwpriv command, which updates the variable ic->random\_gen\_mode. In the kthread, the variable is monitored and the mode can be changed accordingly. All the implementation is placed under macro “ATH\_GEN\_RANDOMNESS”. By default, the mode is set to 0. This implementation helps to update the random number, even in scenarios in which the RSSI is constant.

Enter the following iwpriv command to set the mode:

```
iwpriv athx random_gen_mode <0-2>
```

Description of modes:

- Mode0—Random is updated by rssi on interrupt.
- Mode1—Random is updated by rssi every 10 ms, and if the same rssi is read for 5 times, it is read from ADC regs and a sleep of 100 ms occurs.
- Mode2—Random is updated by analog-to-digital (ADC) registers alone every 100 ms.

If a mode value that is outside of the prescribed range of mode values is entered, the entry is not validated successfully and the setting is not saved.

# 14 Channel Selection

---

This chapter describes the channel selection methodologies and techniques, such as Channel Width Management, Dynamic Frequency Selection (DFS), EACS PLUS: ACS/OBSS, ICM, UMAC, 80 MHz Scan, Dynamic Channel Selection-Interference Mitigation (DCS-IM) for 802.11n and 802.11a, and enhanced channel switch and channel scan API.

## 14.1 Channel Width Management (CWM 20/40)

Channel Width Management is the algorithm used by the WLAN software to choose the channel width (20 MHz or 40 MHz) to use when transmitting a frame to a STA.

### 14.1.1 Theory of Operation

The WLAN hardware supports operation in both 20 MHz and 40 MHz channel widths. Both the PHY and the MAC can independently be set to work in different modes and can operate in different modes. The PHY supports the following two modes:

- Static 20 MHz mode:

In this mode the synthesizer clock is set at the center of a 20 MHz channel. For example, on channel 6 in HT20 mode, the synthesizer is set to 2452 MHz and the channel is from 2442MHz to 2462 MHz. The hardware can transmit and receive on 20 MHz frames

- Dynamic HT20/HT40 mode:

In this mode, the synthesizer clock is set to the center of the 40 MHz channel. For example, when using channel 6 as the primary channel, and the 20 MHZ channel above channel 6 as the secondary channel, the synthesizer is set to 2462 MHz and the 40 MHz channel extends from 2442 MHz to 2482MHz (assuming that the channel spacing is set to 20 MHz). The hardware supports a mode where channel spacing is set to 25 MHz, in which case the channel bandwidth extends from 2442 MHz to 2487MHz. In Dynamic HT20/HT40 mode the WLAN hardware can transmit and receive both 20 MHz and 40 MHz frames. However, 20 MHz frames can be transmitted and received only on the primary channel of the 40 MHz channel.

The MAC can be configured in 20 MHz mode or in 2040 mode. In 2040 mode, the MAC does a Clear Channel Assessment (CCA); that is, the detection of a transmitter on the medium on both the Primary Channel and Secondary Channel. Both the MAC and PHY must be configured together to allow the software CWM State Machine to operate.

The WLAN hardware can operate in one of the modes described above (static HT20 or Dynamic HT2040). When doing a CWM mode change, software changes both the MAC registers as well as

the PHY registers. A Virtual AP (VAP) will use hardware settings. Hence all VAPs on an AP will all use the same hardware settings and any PHY/MAC change impacts all the VAPs.

There are three possible events that can trigger a mode change:

- HT40 intolerant-STA joins/leaves
- STA reported, beacon heard
- Secondary channel sensing (busy or clear)

The events can trigger the CWM Finite State machine through the timer and events

- Timer:

The timer periodically checks the percentage of time that the extension channel is busy and decides on whether the state change is needed.

- Events:

On detecting the need for a state change, an event is queued on the event queue. The events of joining/leaving/reporting of HT40-Intolerant STAs or APs result in events being queued on the event queue.

## 14.1.2 Implementation

### 14.1.2.1 Channel Width Management Data

The main data structure for Channel Width Management is `ath_cwm`. A single instance of this structure is created at attach time. This structure stores parameters such as

- current mode
- the channel offset (+ or -)
- the hardware state (MAC and PHY mode)
- timers and an event queue for the CWM events
- CWM state machine state

### 14.1.2.2 Channel Width Management Functions

The functions can be grouped into the following categories:

- UMAC:
  - Management Layer (UMAC) functions
- LMAC:
  - Attach/detach functions
  - Initialization functions
  - Start/stop functions
  - Scan functions

- Switching functions
- Finite State Machine (FSM) functions

### **Management layer functions: ieee80211\_recv\_action()**

When the AP receives an 802.11 action frame from the wireless network, the function ieee80211\_recv\_action() is called. This function checks to see if the action frame is of the PUBLIC subtype and the action is 0 (HT40Intolerant), or if the action frame is of the HT subtype and the action is a channel width change. If so, it calls the Channel Width Management state machine changes the channel width.

When a new STA associates with the AP, the AP checks to see if the STA is a HT40Intolerant STA. If it is, and if the AP is operating in HT40 mode, it calls the CWM State Machine functions to switch to HT20 mode.

When a STA disassociates with the AP, the AP checks to see if the STA is a HT40 Intolerant STAe. If it is, and if the AP is operating in HT20 mode, it calls the CWM state machine functions to switch to HT40 mode.

### **Attach functions: ath\_cwm\_attach, ath\_dcwm\_detach, cwm\_attach, cwm\_detach**

These functions are in the files ath\_cwm.c and ath\_cwm\_project.c. T

The functions in ath-cwm\_project.c (ath\_cwm\_attach, ath\_cwm\_detach) are the external interface to the CWM module and are wrappers around the functions in ath\_cwm.c (cwm-attach, cwm\_detach).

- ath\_cwm\_attach is called from the ath\_attach function
- ath\_cwm\_attach calls cwm\_attach
- ath\_cwm\_detach is called from the ath\_detach function
- ath\_cwm\_attach calls cwm\_detach

These functions set up the external interface by setting function pointers to functions that get/set various CWM parameters and channel width switching functions. The CWM parameters are set to their default values and also stored in a hardware state data structure.

Following are the important parameters:

- CWM Mode: The default is HT20
- Extension offset: The default is 0; that is, the lower 20 MHz channel
- Extension mode: The default is none
- Extension spacing: The default is 25 MHz

The detach functions free any memory associated with the CWM state machine. In addition they also cancel any timers and remove any events in the event queue.

### **Initialization functions: cwm\_init, cwm\_hwinit()**

These functions (in ath\_cwm.c and ath\_cwm\_project.c) initialize the CWM internal variables. The variables are divided into two categories: CWM state machine variables and hardware state

variables. These functions are called from `ath_init()`. The CWM state machine includes the parameters described in the previous section.

- `cwm_init()`: This function takes the channel mode from the `cw_struct` (set during channel selection) and sets the CW parameters accordingly.
- `cwm_hwinit()`: This function initializes the hardware parameters in the `cw_struct`

### **Start/Stop functions: `ath_cwm_up`, `ath_cwm_join`, `ath_cwm_down`, `cwm_up`, `cwm_down`, `cwm_start`**

The functions (in `ath_cwm.c` and `ath_cwm_project.c`), start and stop the CWM state machine.

`ath_cwm_up` is called for each VAP after the channel is set. It is also called from `vap_up`. It in turn calls `cwm_up` which calls `cwm_start`

`cwm_start` checks to see if the CWM state machine is already running, if the hardware is not in HT2040 mode, or if CWM should be disabled by regulatory requirements. If none of these are true, the function initializes the CWM state machine and starts the CWM timer.

The CWM timer is a timer function that expires at fixed intervals and causes the CWM state machine to process CWM events in its event queue.

`ath_cwm_join` is called from `vap_join`. It in turn calls `cwm_join`. This function sets the CWM parameters for the VAP and initializes the hardware state for the CWM state variables.

`ath_cwm_down` is called from `vap_down`. It calls `cwm_down`, which calls `cwm_stop`, which in turn cancels the CWM timer, thus stopping any CWM events from being processed.

### **Scan functions: `Ath_cwm_scan_start`, `Ath_cwm_scan_end`**

`ath_cwm_scan_start()` stops the CWM state machine during a scan. It iterates through all the VAPs, passing the scan function `ath_cwm_scan()` as the parameters. `ath_cwm_scan()` in turn calls `cwm_scan()`, which stops the CWM state machine through `cwm_stop`.

`ath_cwm_scan_end()` iterates through all the VAPs, passing the scan function `ath_cwm_join()` as the parameter. The `ath_cwm_join()` function restarts the CWM state machine.

### **CWM switching functions: `ath_cwm_switch_mode_dynamic2040`, `ath_cwm_switch_mode_static20`, `ath_cwm_macmode`, `ath_cwm_chwidth_change`**

These functions are used to change the MAC and PHY CWM modes. `ath_cwm_switch_mode_dynamic2040` first calls `cwm_switch_mode_dynamic2040`, which in turn calls the CWM MAC and PHY functions.

CWM PHY functions: The `cwm_action_phy40to20` first sets the device channel flags. It then sends an action management frame to inform STAs of the change and calls the UMAC channel change functions. There is a corresponding 20to40 function.

CWM MAC functions: The `cwm_action_mac40to20` function first sets the MAC mode to 20 MHz. It updates the rate control structures for all nodes, stops the Tx DMA, processes completed Tx frames, requeues frames, and restarts Tx DMA. It then sends an action management frame to inform STAs of the change.

### CWM Finite State Machine functions: cwm\_timer, cwm\_queueevent

Timer function: cwm\_timer(): When the CWM timer elapses the CWM FSM collects the extension channel busy percentage (using cwm\_extchbusy()). It uses the results to calculate the extension channel state (CLEAR or BUSY).

If current state is CLEAR and the new channel state is BUSY, it queues an Extension Channel Busy event on the CWM event queue.

If current state is BUSY and the new channel state is BUSY and the state has been persistent for at least one timer interval, it queues an Extension Channel Stop event,

If current state is BUSY and if the new channel state is CLEAR, it queues an Extension Channel Clear Event.

If channel is UNAVAILABLE and it has been so for a persistent interval, it queues is Extension Channel Resume event

Event processing function: cmw\_queueevent(): This function queues the event in the CWM event queue. It then attempts to acquire ownership of the queue. If it does, it traverses the queue and calls the state transition function for every entry in the queue. The state transition functions call the appropriate MAC and PHY CWM mode change functions are described in the previous section.

#### 14.1.3 API

The CWM API is provided by the functions in ath\_cwm\_project.c The functions are explained in the preceding section. The list is provided below:

- void ath\_cwm\_up(void \*arg, struct ieee80211vap \*vap);
- void ath\_cwm\_down(struct ieee80211vap \*vap);
- void ath\_cwm\_join(void \*arg, struct ieee80211vap \*vap);
- void ath\_cwm\_scan\_start(struct ieee80211com \*ic);
- void ath\_cwm\_scan\_end(struct ieee80211com \*ic);
- void ath\_cwm\_chwidth\_change(struct ieee80211\_node \*ni);
- void ath\_cwm\_switch\_mode\_static20(struct ieee80211com \*ic);
- void ath\_cwm\_switch\_mode\_dynamic2040(struct ieee80211com \*ic);
- void ath\_cwm\_switch\_to40(struct ieee80211com \*ic);
- void ath\_cwm\_switch\_to20(struct ieee80211com \*ic);

#### 14.1.4 Configuration

There are no CLI commands to change the default Channel Width Management operations.

### 14.1.5 Offload implementation

The offload does not support CWM on its own, as the device with offload architecture is supporting it in the hardware.

QUALCOMM®  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

## 14.2 Dynamic Frequency Selection (DFS)

Dynamic Frequency Selection (DFS) is a mechanism that allows unlicensed devices to use the 5 GHz frequency bands already allocated to RADAR systems without causing interference to those RADARs. The concept of DFS is to have the unlicensed device detect the presence of a RADAR system on the channel they are using and, if the level of the RADAR is above a certain threshold, vacate that channel and select an alternate channel.

The objectives of DFS are:

1. Coexist in 5 GHz channels with RADAR
2. Perform uniform loading or spreading across the spectrum. There are many channels in the 5 GHz bandwidth in which wireless device can operate. DFS also requires random channel selection so that channels are uniformly utilized across the spectrum.

Compliant operation in the 5 GHz band requires that access point equipment detect the presence of in-band RADAR signals. Generally these signals take the form of periodic narrow band pulses of length 1 – 100 µs, or chirp pulses of 50 – 100 µs. In the event of a RADAR detection, the access point is required to vacate the RADAR co-existed channel (within 10 s), and hop to a different channel. This section of the document describes the DFS software implementation

For more detailed descriptions of the specific regulatory requirements, please refer to the relevant documents from IEEE (802.11h), ETSI, ITU and other regulatory bodies.

### 14.2.1 Terminology

The following are terms used by DFS.

**Channel Availability Check Time:** The time a system shall monitor a channel for the presence of RADAR prior to initiating a communications link on that channel. This is also referred to by the acronym CAC.

**Interference Detection Threshold:** The minimum signal level, assuming a 0 dBi antenna, that can be detected by the system to trigger the move to another channel.

**Channel Move Time:** The time for the system to clear the channel and measured from the end of the RADAR burst to the end of the final transmission on the channel.

**Channel Closing Transmission Time:** The total, or aggregate, transmission time from the system during the channel move time.

**Non-Occupancy Time:** A period of time after RADAR is detected on a channel that the channel may not be used.

**Master Device:** Device that has RADAR detection capabilities and can control other devices in the network (for example, an Access Point would be considered a master device)

**Client Device:** Device that does not initiate communications on a channel without authorization from a master device (for example, a laptop Wi-Fi card – note that a Wi-Fi card that supports ad-hoc mode would be considered a master device)

## 14.2.2 Theory of Operation

The operation of a system with DFS capability takes the following sequence (see Figure 14-1):

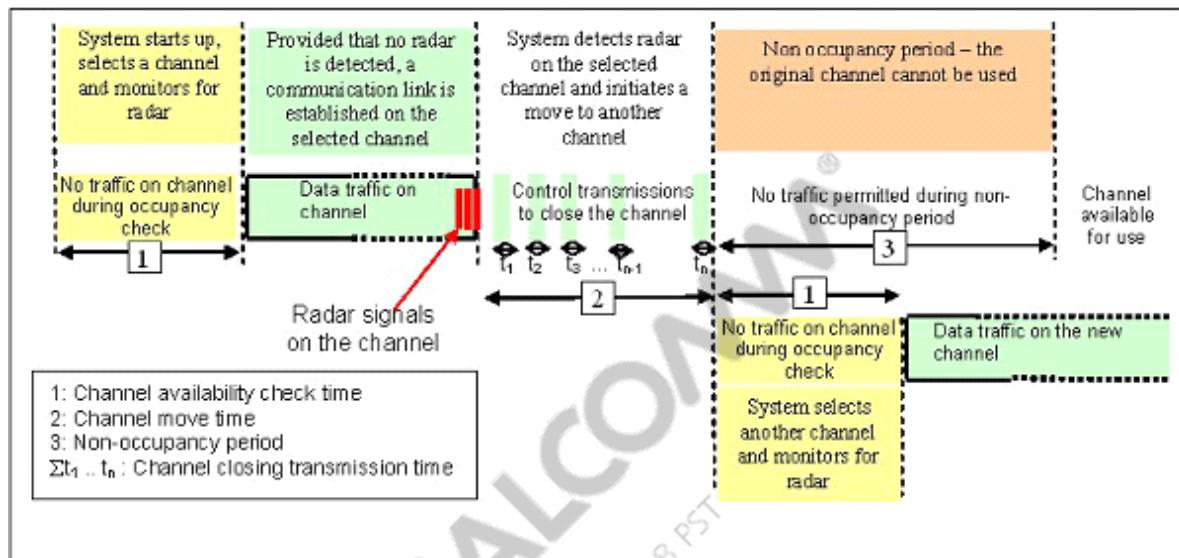


Figure 14-1 DFS Overview

The master device selects a channel and if it is a DFS channel, the master would start the CAC period to monitor that channel for potential RADAR interference for a minimum listening time (channel availability check time). No transmissions can occur during this period. If interference is detected, then the system has to select another channel and repeat the channel availability check on the new channel (the original channel is added to a list of channels with RADAR).

Once a channel has been selected and passes the channel availability check, the network starts to use that channel.

While using the channel, the network master device continuously monitors for potential interference from a RADAR source (referred to as in-service monitoring). If interference is detected, the network master device issues commands to all other in-network devices to cease transmissions. The channel is added to the list of channels with RADAR and the master device then selects a new channel (one that is not on the list). The sequence starts again with a channel availability check.

A channel on the RADAR list can be purged once the non-occupancy period has elapsed for that channel.

Regulatory requirements for DFS may vary from country to country, but the fundamentals are the same.

### 14.2.3 Implementation

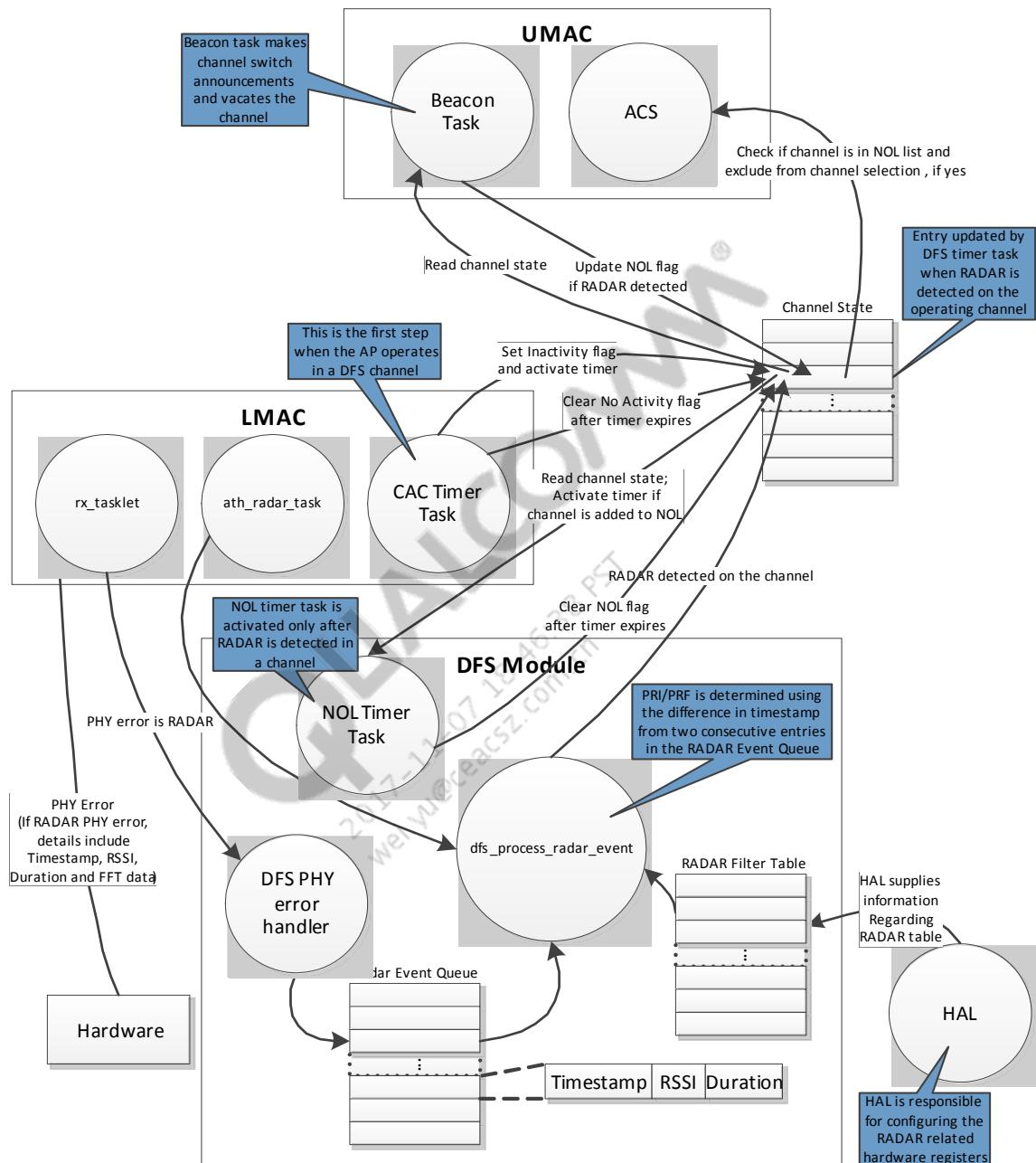
A dedicated RADAR detection block in Qualcomm Technologies chips is responsible for distinguishing RADAR pulses from incoming beacon/management or data frames.

Whenever the hardware detects a RADAR PHY error, it generates an interrupt and the MAC layer provides the following information to software (interrupt handler):

- RSSI value
- Pulse width
- Timestamp on the collected data. This information is used to determine the PRF/PRI value by comparing time stamps on two consecutive pulses.
- FFT data that is analyzed for whether it is a long RADAR pulse. A RADAR waveform is identified by the following:
  - Repetition frequency/pulse repetition interval (PRF/PRI)
  - Chirping pulse, where PRF/PRI may vary
  - Strength of the pulse (RSSI)
  - Width of the pulse

There is an event queue associated with RADAR detection. The RSSI, pulse width, and time stamp is stored in the queue by the interrupt handler.

There is a DFS timer task that reads the entries in the queue associated with RADAR detection. It checks the time stamp, RSSI, and pulse width value against the PRI, RSSI and pulse width defined in the RADAR filter table. This is illustrated in [Figure 14-2](#).

**Figure 14-2 DFS Operation**

### 14.2.3.1 Slave DFS implementation

A slave device (Client/STA device) whose maximum Tx power is greater than or equal to 200 mW has to follow DFS. In the current implementation the driver does not automatically detect it, if the maximum Tx power for the client is greater than or equal to 200 mW.

The following command enables or disables client/STA mode DFS.

```
iwpriv wifiX staDFSEnable 0/1
```

The theory of operation of DFS is same for both client/STA and AP mode with the following exceptions:

- STA enables radar detection only after the STA is connected to the Root- AP.
- Once RADAR is detected in a channel it is added to a list called NOL-History list. This is different from the NOL list. No channel is removed from the NOL-History list until driver is unloaded. When the NOL expires, the channel is removed from the NOL list but the channel will continue to remain in the NOL-History list.
- The STA performs Channel Availability Check (CAC) before joining the AP. If the NOL-History flag is set for the channel. i.e., if the STA is going to be associated with a RADAR channel for the first time, it does not perform CAC. It performs CAC, for the second and subsequent associations if it has already detected RADAR in that channel. The channels in the NOL-History can be listed by executing the command ‘radartool -i wifiX shownolhistory’.
- STA CAC is done only for the countries that are in the ETSI domain.
- After radar detection STA does not send any Channel-Switch-Announcement (CSA).

### 14.2.3.2 Repeater DFS implementation

- When the RootAP detects a Radar it chooses a random channel and sends CSA to connected clients. It adds Radar channel to NOL so that the RootAP will not use that channel for 30 minutes.
- When the RepeaterAP receives a Non-DFS-CSA\* from the RootAP, the RepeaterAP VAPs propagate the RootAP CSA to its BSS. Both STA and AP vap come up in Root AP's channel.
- If the RepeaterAP receives a DFS-CSA\* from the Root AP or if it detects a local radar, it does a random channel selection from Non-DFS channel list, if no Non-DFS channel is available then it chooses a DFS channel. The RepeaterAP's AP VAPs send local CSA (with new channel) to their BSSes. The Repeater STA vap gets beacon miss and disconnected from the Root AP.
- If the Repeater is in Independent mode:
  - The Repeater AP VAPs do a CAC if the new channel is DFS and after CAC start beacons in new channel. The STA VAP starts scanning.
  - If the Repeater STA finds the RootAP in Non-DFS channel, then the RepeaterAP vaps starts local CSA (New channel is RootAP's channel). Both the RepeaterAP's AP VAPs and the STA VAP come up in new channel.
  - If the Root AP is in DFS channel and the Repeater AP is not in same channel, the Repeater AP VAPs start local CSA with the new DFS channel and do a CAC. After CAC the

Repeater AP VAPs start beaconing in new channel and the STA vap connects to the RootAP.

- If the Repeater detects a Radar during CAC, it marks the channel as Radar and adds it to NOL. The Repeater AP vap comes up in Non-DFS channel and the STA vap gets disconnected and starts scanning.
- In dependent mode:
  - The Repeater STA vap brings down the AP VAPs and starts scanning.
  - If the Root AP found in Non-DFS channel, the STA vap gets connects to the RootAP and brings up the AP VAPs. The Repeater AP VAPs starts beaconing in new channel.
  - If the RootAP channel is DFS, the Repeater STA vap does CAC and connects to the Root AP and then brings up the AP VAPs
  - If the Repeater STA vap detects a radar during CAC then it adds the channel to the NOL, cancels the CAC timer and starts scanning.

\* DFS-CSA = In the Channel switch announcement IE the new channel is DFS.

\* Non-DFS-CSA = In the Channel switch announcement IE the new channel is non-DFS.

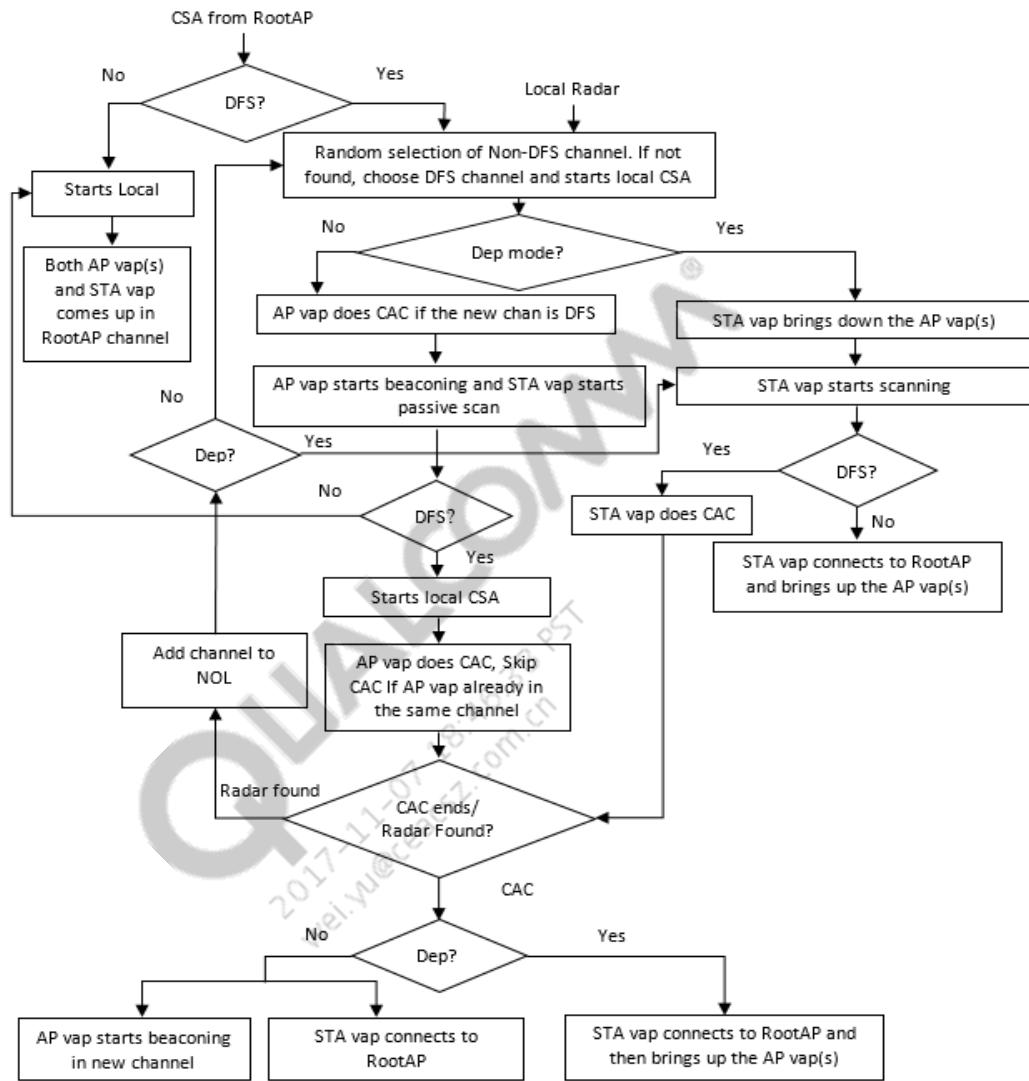
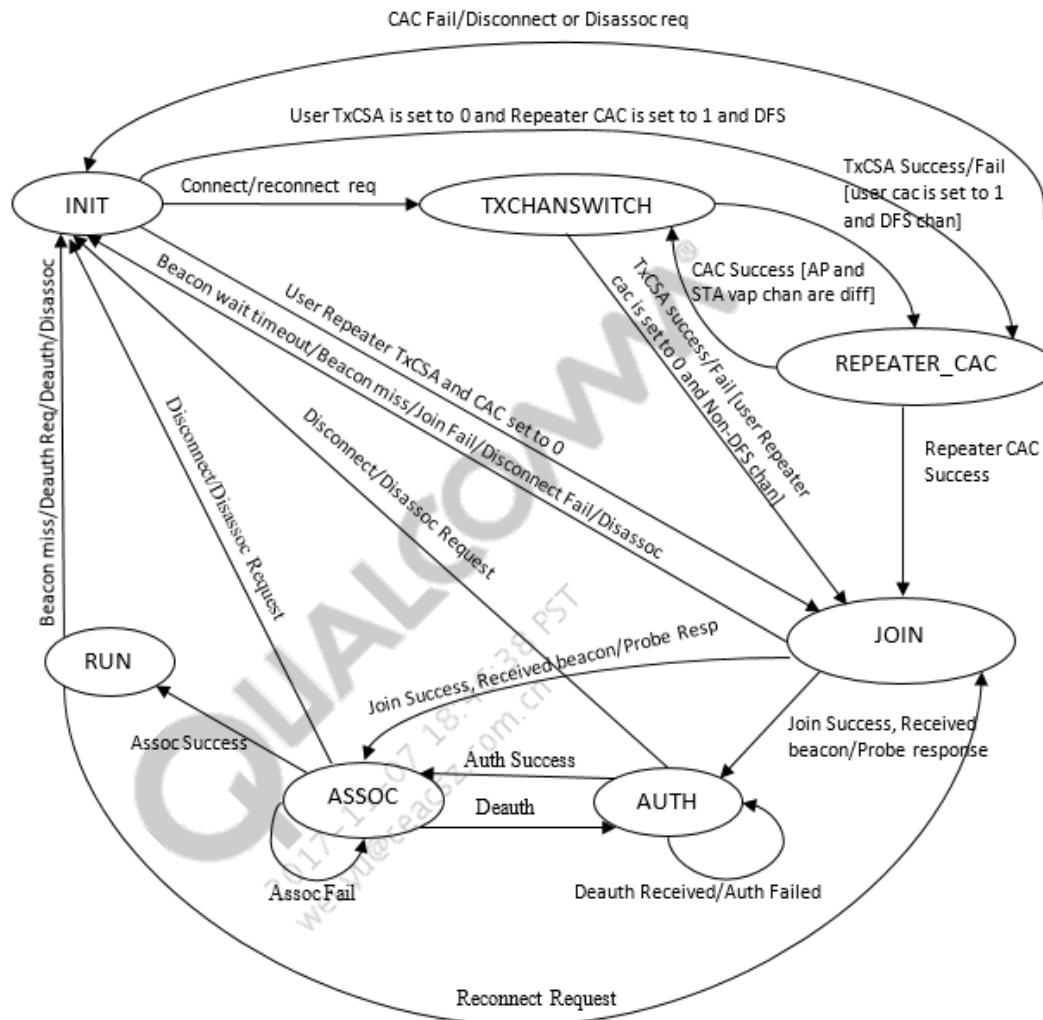


Figure 14-3 CSA and Local Radar handling in Repeater AP



**Figure 14-4 Modified Assoc state machine to support Repeater DFS**

#### 14.2.3.3 Enhanced DFS support for RE to inform Root AP of radar detection

If only the Repeater (RE) detects RADAR but the ROOT does not then RE switches to new channel (Channel B) but the ROOT continues to use the old (channel A) channel. Since the RE adds channel A to its NOL list, RE cannot scan/use channel A for next 30 minutes. This causes a break in the connection between the ROOT and the Repeater for next 30 minutes.

If an RE detects RADAR and has an uplink connection then it sends a message (vendor/proprietary) to the ROOT that RE has detected RADAR and need to vacate the channel. Upon receiving the message, the ROOT chooses a new random channel and sends CSA to its BSS. All the nodes/devices in the BSS switches to the channel mentioned in the CSA. The new message (vendor/proprietary) is called RCSA (reverse CSA). This is a vendor specific action frame sent by a Repeater to its ROOT. For every RADAR detect, 5 RCSA action frames are sent.

The interval between 2 RCSA frames is 100ms (beacon interval). In case of QCA9984 (chain) of repeaters/REs, an RE can both send and receive RCSA. If an RE detects RADAR it sends RCSA to its uplink/ROOT which itself can be an RE.

If Repeater detects radar but not ROOT, Repeater informs Root about the radar detection through RCSAs. Repeater could wait for CSA from Root to switch to next channel. In above approach, a fixed timer is used assuming maximum hop count. However, one should vacate the channel as soon as possible after the radar is detected.

Therefore, to move to a new channel (random) immediately after sending RCSAs, one should know the next channel before hand. This is handled by a Root which can propagate apriori random next channel info in all beacons. When Repeaters connect to ROOT they copy and propagate this info to their BSS. This is performed through a proprietary VENDOR IE.

This way the full network comprising Root and cascaded repeaters are informed of the next channel to move, upon radar detection. Once radar is detected in repeater, it informs ROOT about the same via RCSA and switch to next channel already advertised. This avoids waiting for CSA from Root.

## Algorithm

- If an RE detects RADAR and has an uplink connection (STA is connected to another AP) then it sends RCSA to uplink and starts 2sec timer because it takes some time for RCSA to reach the ROOT and then the ROOT to send CSA. Also the propagation of CSA in case of QCA9984 of REs takes some time. Therefore, the RE that detects the RADAR has to wait for some time for the CSA to arrive. Assuming there are a max of 5 REs in QCA9984 it takes 500ms for RCSA to reach the ROOT and 500ms for the CSA to reach the last terminal RE, RE need to wait for 1 sec before it decides that it has waited long enough for the CSA to come. RE set a 2 sec (500 ms+500 ms+1 sec (grace period)) timer in each RE that sends the RCSA. If the timer expires then the RE chooses a random channel and switches to that channel.
- If an RE receives RCSA and has an uplink connection (STA is connected to another AP) then it sends RCSA to uplink.
- If a ROOT receives RCSA, it sends CSA to its BSS. (An RE without uplink connection behaves like a ROOT).
- An RE does not receive any CSA within 2sec then it chooses a random channel and switches to that channel.
- Alternatively, Root can share apriori random next channel to its BSS through a special vendor IE in beacon. Once RE detects radar, it sends RCSAs to its parent and switch to the random next channel communicated earlier. RE need not wait for CSA from its parent to switch channel when this mechanism is used.

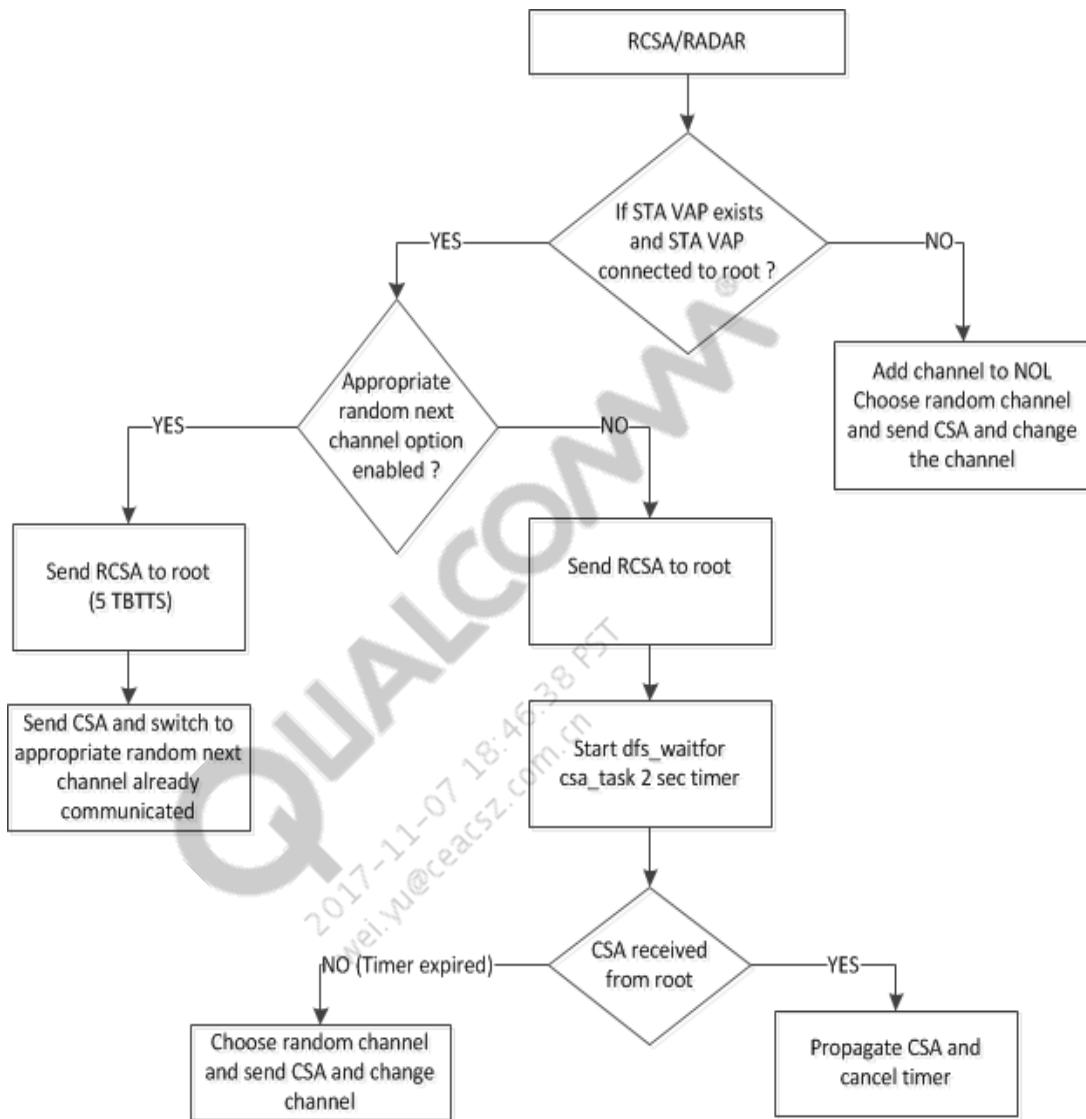


Figure 14-5 Enhanced DFS support for RE to inform Root AP of radar detection

#### 14.2.3.4 Zero CAC DFS

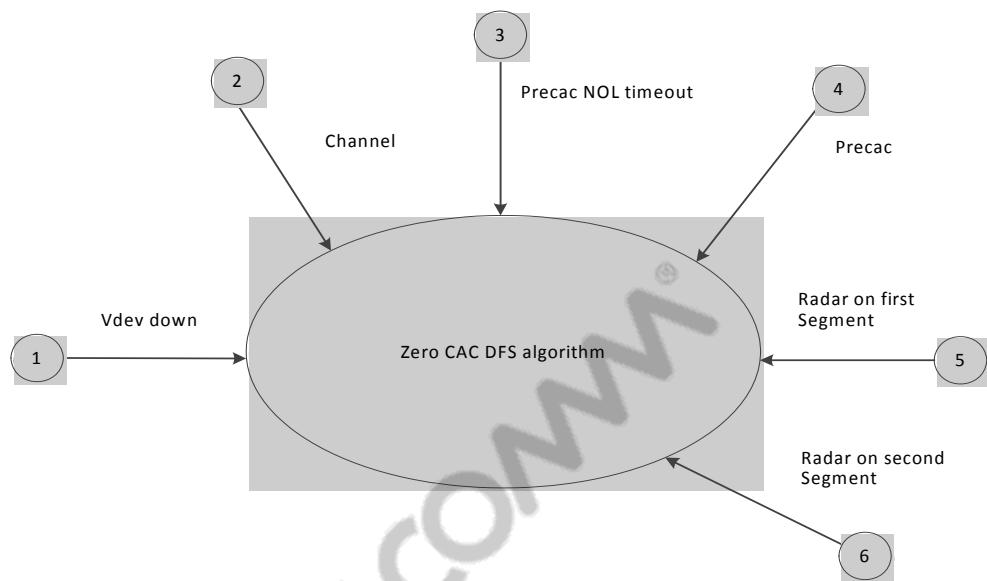
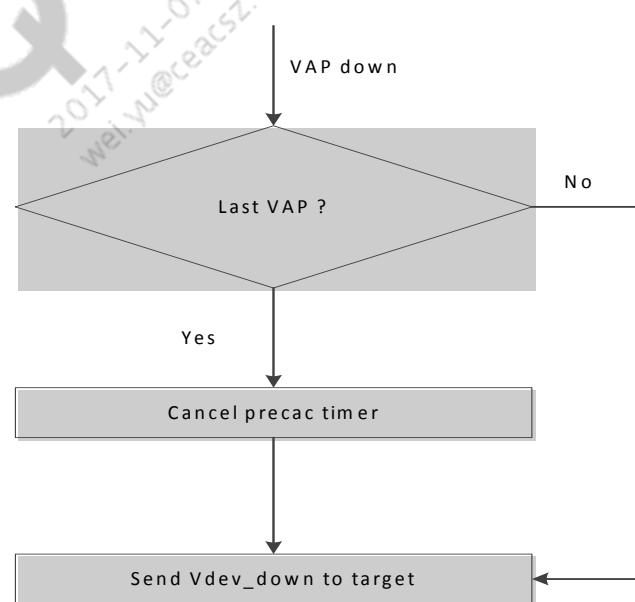
| Acronym                 | Definition  |
|-------------------------|---|
| Agile Mode              | Uses a radio or a PHY/DFS engine to detect radar only.  |
| Pre-CAC                 | performs CAC before using the channel. This is valid only in ETSI and is not allowed in FCC.  |
| Pre-CAC required list   | This is a list of channels that lists all the channels for which CAC is not performed. All VHT80 DFS channels are added to this list during initialization. |
| Pre-CAC Done List       | This is a list of VHT80 channels for which CAC is performed. During initialization this list has no entry.  |
| Pre-CAC NOL List        | This is a list of VHT80 channels for which RADAR is found. During initialization this list has no entry.  |
| Pre-CAC Timer Task      | This timer is used provide a mechanism stay in a channel for the required CAC duration and move on to a new channel at the end of CAC.                      |
| RADAR on First Segment  | If the hardware is in VHT80_80/VHT160 and RADAR gets detected in Primary VHT80 then it is RADAR on first segment.   |
| RADAR on Second Segment | If the hardware is in VHT80_80/VHT160 and RADAR gets detected in secondary VHT80 then it is RADAR on second segment.  |

Operation in a DFS channel requires CAC that adds additional delay as well as loss of connection even when CSA is used. ETSI allows pre-CAC, i.e. performing CAC at a convenient time and using that channel later. Once the Pre-CAC is done in a channel, it is no longer required to perform a CAC in the channel before TX/RX as long as radar is not found in it or we reset or restart the device. This feature applies only to QCA9984 and its specific PHY mode (VHT80\_80, VHT160). It will use only 80MHz primary channel for receive and transmit operation. This is not available in VHT20, VHT40 or VHT80 mode with second PHY/DFS disabled. It is also limited to ETSI domain.

In a system with VHT80\_80 mode, an AP uses primary VHT80 for normal TX, RX, and Radar detection (if the channel is DFS). At the same time the secondary VHT80 can perform CAC in another DFS channel. For example, the AP may set the primary VHT80 segment to channel 52 to perform CAC followed by transmit/receive/radar monitoring operation and may set the secondary VHT80 segment to channel 100 to perform CAC only (no Tx, Rx). The AP may also set the primary VHT80 segment to a non-DFS channel like 36. The AP sets the secondary VHT80 segment to a DFS channel and listen to the channel for CAC duration. If no radar is detected during that time, it considers that pre-CAC for the channel successfully finished. AP will switch the channel of the secondary segment to another DFS channel and repeat the operation. By using this method AP will prepare a list of channels where CAC is already done (Pre-CAC done list). Before using a channel AP will check the Pre-CAC done list to see if CAC is required in the channel. If the channel is present in the Pre-CAC done list then no CAC will be done.

A precac\_timer task is used to move from one channel to other for the secondary VHT80 segment. If RADAR is found during CAC the VHT80 channel is added to the Pre-CAC NOL list. The Pre-CAC NOL list behaves like a regular NOL list except that it contains only the VH80 channels.

Zero CAC DFS events:- The Zero CAC DFS algorithm runs in response to the following events show in the Figure.

**Figure 14-6 Zero CAC DFS Events****VAP down****Figure 14-7 Vap down**

## Channel change

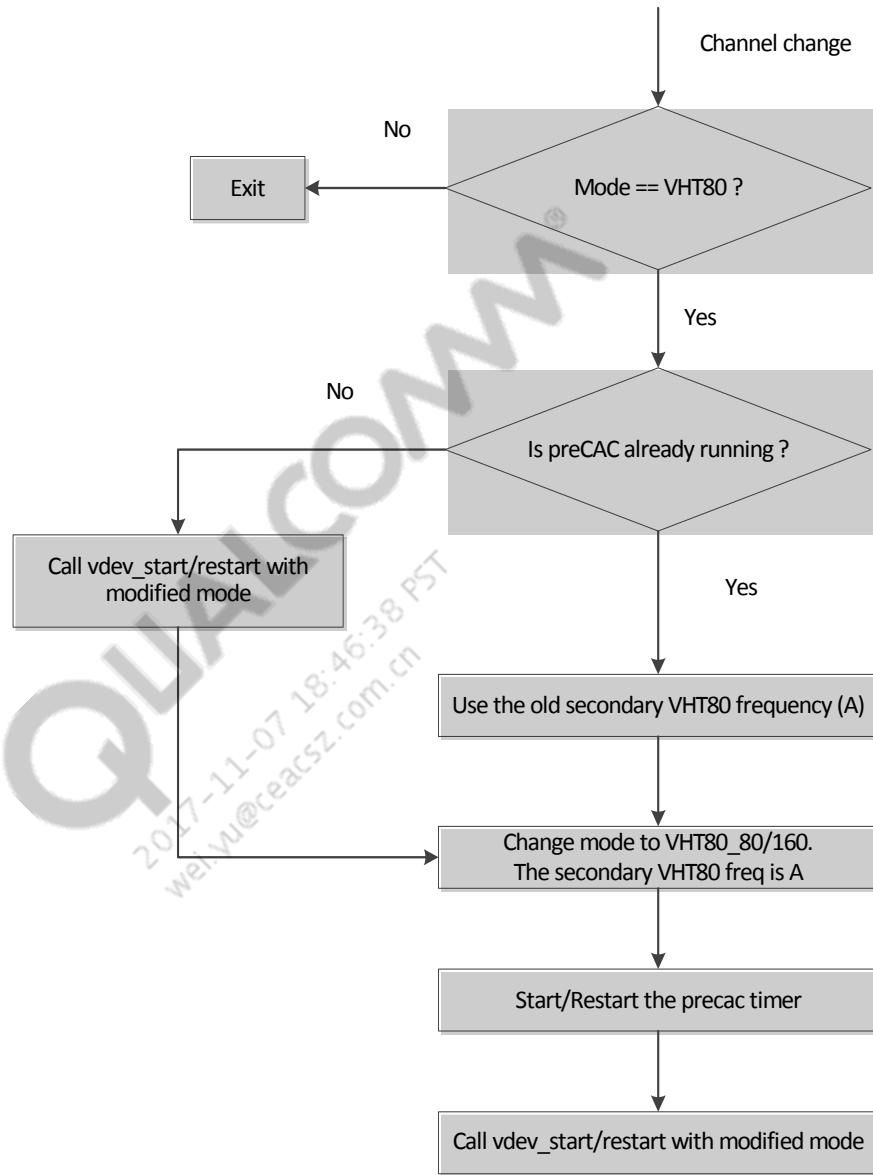


Figure 14-8 Channel change

### PreCAC NOL timeout

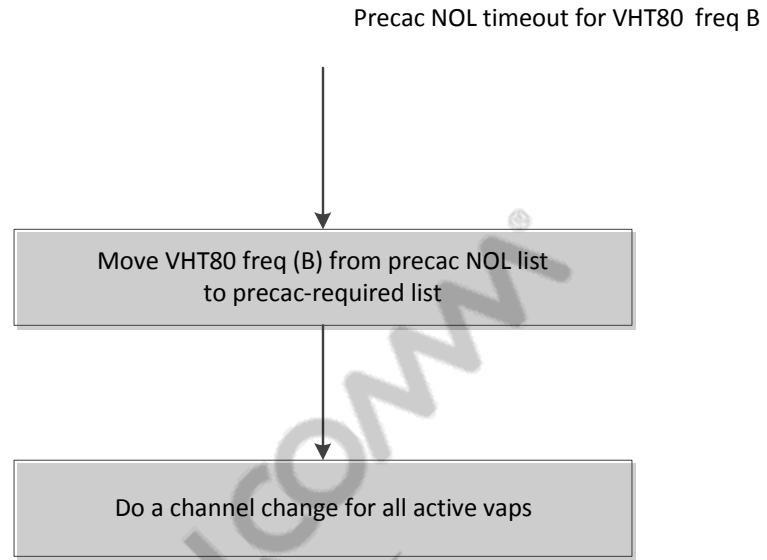


Figure 14-9 PreCAC NOL timeout

### PreCAC Timeout

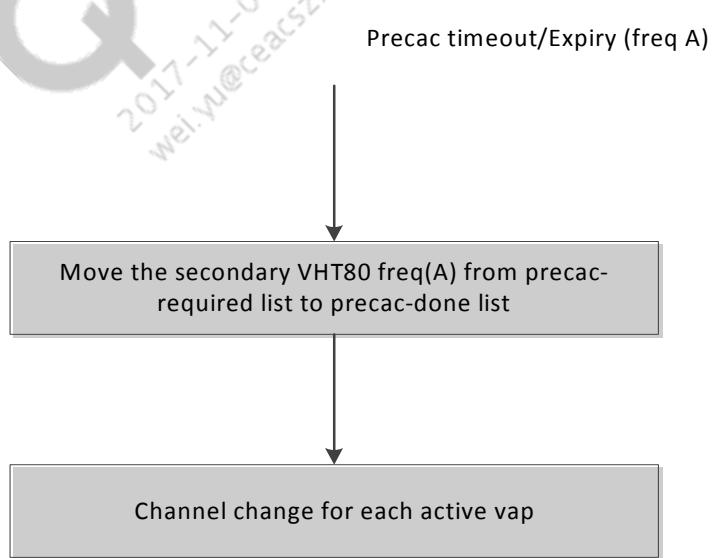
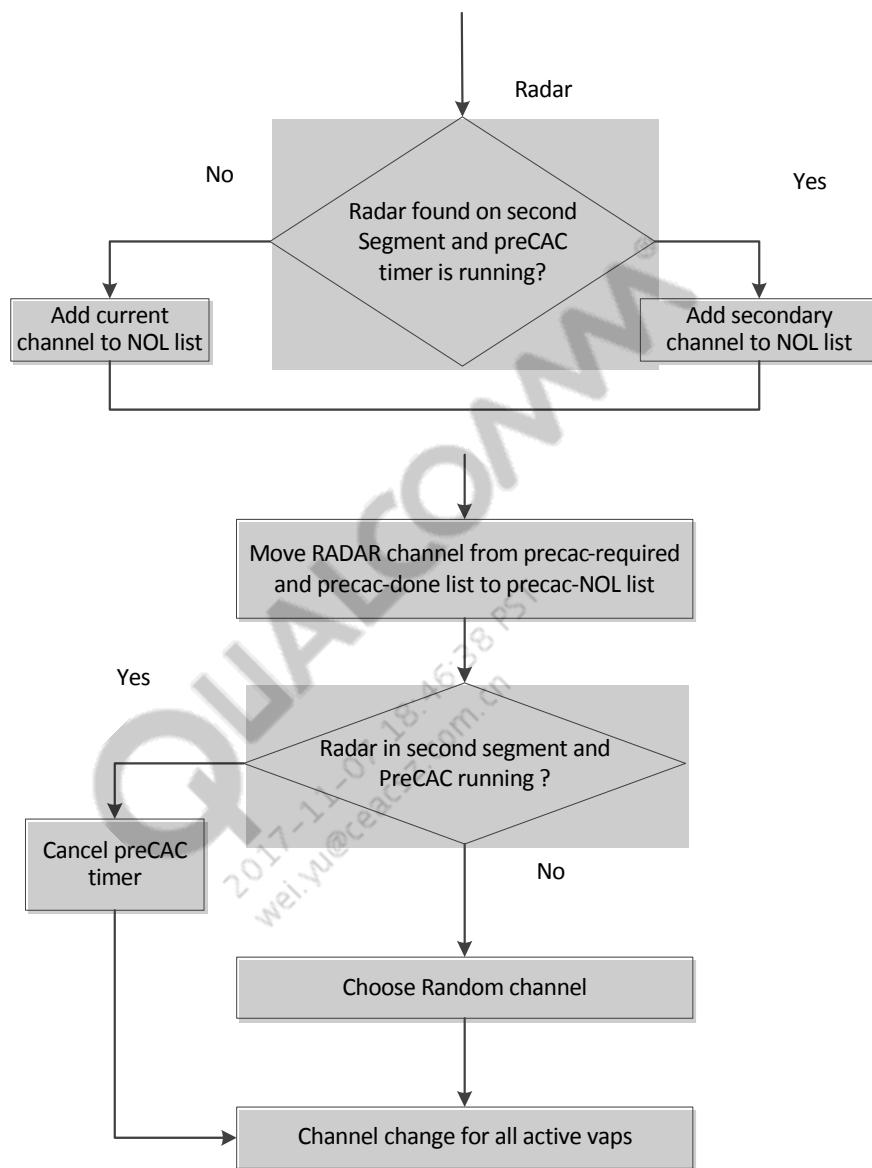


Figure 14-10 PreCAC Timeout

### Radar detection on first and second Segment



**Figure 14-11 Radar detection on first and second Segment**

**NOTE** When a radio switches to channel 132, 36, and 140, pre-CAC functionality is disabled and no configuration option is available to resume the feature. The AP has been configured in VHT80. When changing channel to 149, the consistency between channel and mode fails. This behavior occurs because 149 does not support VHT80. As a result, the code falls back to auto mode, which is VHT20. This is an expected behavior. The administrator must know the modes supported for different channels.

### 14.2.3.5 Codebase

This section explains the various files that make up the DFS module. The files are split up by functionality to make the module easier to maintain.

- **dfs.c**

This was the large original file that has now been split up into multiple files. It contains the `dfs_attach()` and `dfs_detach()` functions as well as the `dfs_control()` function which is used to process ioctls related to DFS. For Linux/Mac, ‘radartool’ is the command line tool that can be used to call various ioctls to set and get RADAR detection thresholds.

- **dfs\_process\_phyerr.c**

For each RADAR pulse that the hardware detects, a single RADAR PHY error is reported to the driver. This PHY error contains information like the RSSI, the pulse duration, the pulse location (primary/extension/DC) and possibly FFT data.

This is the file that contains the functionality to process the RADAR PHY error that is reported by the chip to the driver. Once `ath_recv` figures out that this is a RADAR PHY error it will call `ath_process_phyerr()` and pass to it the descriptor of the PHY error. A very basic level of filtering is done on the PHY error and a RADAR event is generated.

- **dfs\_process\_radarevent.c**

This contains the functionality to process the RADAR event generated for a pulse. This will group together pulses and call various detection functions to figure out whether a valid RADAR has been detected.

- **dfs\_bindetects.c**

DFS specs specify the various types of RADARs to be detected. Each separate type is called a Bin and has different characteristics. This file contains the functionality to look at a group of pulses and to detect whether we have detected a valid RADAR waveform. To do that, it must match the group against the different characteristics of each Bin.

- **dfs\_fcc\_bin5.c**

FCC Bin5 is a special type of RADAR because they “chirp”. Basically the pulses move across the frequency band and are called chirping pulses. `dfs_check_chirping()` examines the FFT data contained in the PHY error information to figure out whether the pulse is moving across frequencies.

- **dfs\_staggered.c**

ETSI 1.5.1 introduced new waveforms which use staggered PRIs within the same waveform. This file contains the detection implementation for these specific types of RADAR signals. This detection logic is different from others because it must detect waveforms that may have 2 or more different PRIs (pulse repetition intervals).

- **dfs\_nol.c**

This contains NOL-related functionality, NOL being the non-occupancy list. After RADAR has been detected in a particular channel, the channel cannot be used for a period of 30 minutes, which is called the non-occupancy. The NOL is basically a list of all the channels on which RADAR has been detected. Each channel has a 30-minute timer associated with it. This file contains the functionality to add a channel to the NOL, the NOL timer function and the functionality to remove a channel from the NOL when its time is up.

- **dfs\_init.c**

This file contains initialization functions and functions that reset internal data structures.

In this file, `dfs_init_radar_filters()` is the function that initializes the RADAR filter tables based on the regulatory domain set. This is called at initialization, and in every instance a country code is set (that is, when a new regulatory domain is set and a different set of filter tables needs to be used)

- **dfs\_misc.c**

This file contains miscellaneous functions that did not fit in anywhere else.

- **dfs\_debug.c**

As is suggested by its name, this file contains useful print functions that can be used for debug. Add all debug-related functionality into this file.

- **dfs\_ar.c**

This file contains now obsolete code which previously implemented the AR (Adaptive Radio) feature for older chipsets. This is kept for historical reasons only and is not used.

- **if\_ath\_dfs.c**

This file provides interface functions for the UMAC to access channel states that are affected by the DFS module.

#### 14.2.3.6 Software APIs

- **dfs\_getchanstate**

For a given channel, this function is used to inquire if it is a Control or Extension channel.

- **dfs\_process\_radarevent**

This is the RADAR timer task. This goes through the RADAR event queue, gets the information regarding the events, and matches it against the information in the RADAR filter table.

- **dfs\_check\_nol**

This function checks to see if a channel is in the No-Occupancy list. When selecting a channel on which to operate, the LMAC will skip a channel if it is in this list.

- **dfs\_attach**

This is the attach function to register various handlers with the Linux OS and to initialize various data structures. Also, any external data or functions that the driver must use are also passed by the caller through this function.

- **dfs\_detach**

This is a cleanup function which has the reverse effect of `dfs_attach`. It frees up all the resources that the module used during its lifetime.

- **ath\_ar\_enable**

This function is no longer in use and is present only for historical reasons.

- **dfs\_radar\_enable**

This function looks at the current DFS domain and if it is a valid domain, enables RADAR detection for it.

- **ath\_ar\_disable**

This function is no longer in use and is present only for historical reasons.

- **ath\_process\_phyerr**

This is the deferred procedure call for the interrupt handler. This is illustrated in [Figure 14-2](#).

- **dfs\_process\_ar\_event**

This function is no longer in use and is present only for historical reasons.

- **dfs\_control**

This is the interface for any user application to interact with the driver. The radartool application uses this interface to control the configuration (for example, usenol) of the driver and also to extract information (such as statistics) from the driver.

- **dfs\_get\_thresholds**

This function reads the hardware value of certain registers by calling the HAL.

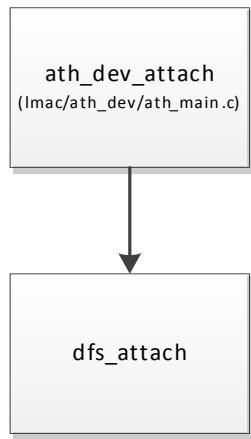
- **dfs\_clear\_stats**

This function resets all statistics that the DFS module has collected to zero.

#### 14.2.3.7 API Flows

##### Attach

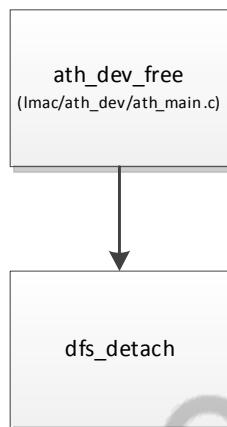
The attach flow is shown in [Figure 14-12](#).



**Figure 14-12 DFS Attach**

## Detach

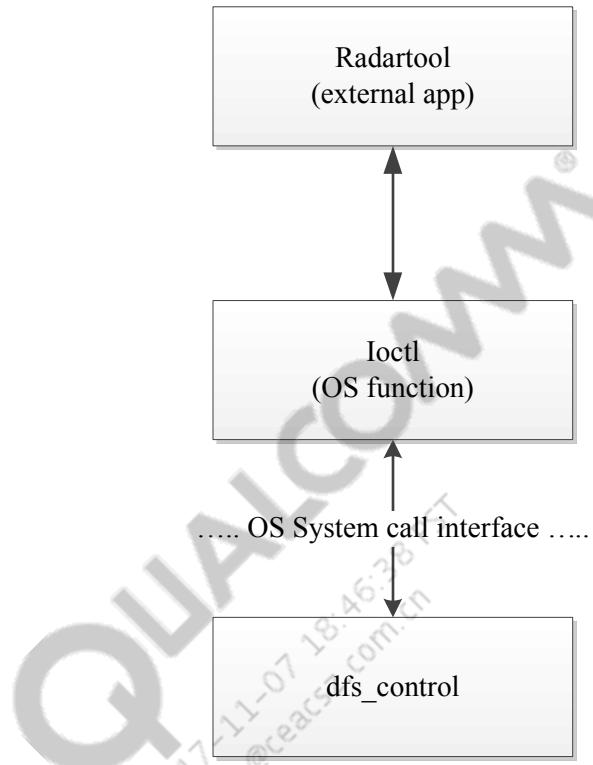
The detach flow is shown in [Figure 14-13](#).



**Figure 14-13 DFS Detach**

### 14.2.3.8 ioctl

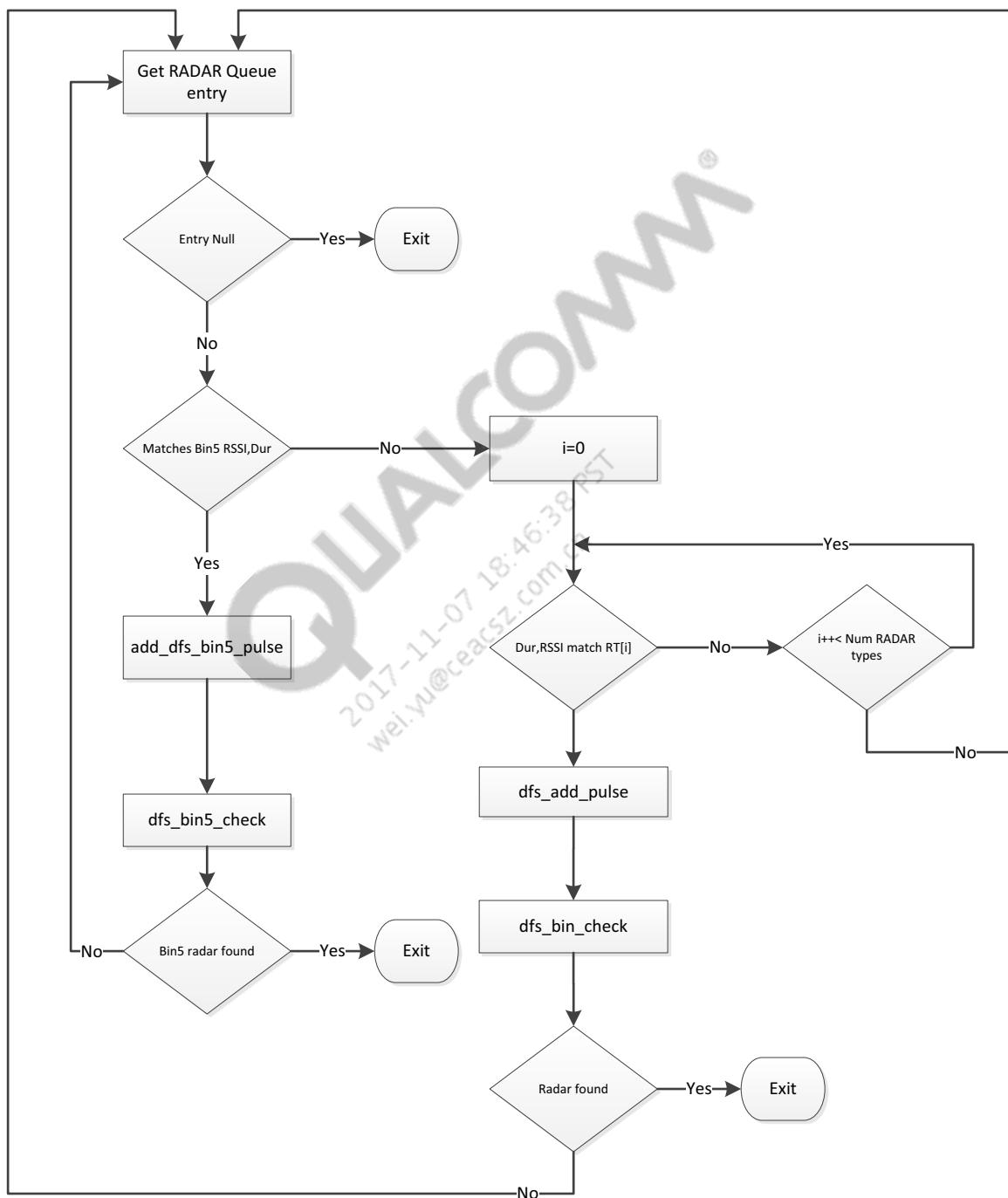
The ioctl flow from a user level application to the driver is shown in [Figure 14-14](#).



**Figure 14-14 ioctl Control**

### 14.2.3.9 Radar Detection Path

Figure 14-15 shows the flow to detect and process RADAR.



**Figure 14-15 DFS Radar Detection Flow**

## 14.2.4 Configuration

There are a few parameters that can be affected during the process of loading the DFS module

- **domainoverride**

This parameter can take one of the following values:

0 = Uninitialized (default)

1 = FCC Domain (FCC3, US)

2 = ETSI Domain (Europe)

3 = Japan Domain

- **usenol**

Setting this parameter to 0 causes the DFS module to not change (vacate) a channel even if RADAR is detected. This is used only for DFS certification and test purposes.

- **countrycode**

This parameter can be used to set the country code. If this parameter is set, this value would be used to automatically select the corresponding DFS domain. Use country code 0x1ff for demo channels.

## 14.2.5 CLI Commands

### 14.2.5.1 Radartool

Radartool can be used to configure the DFS module in ways that help with debugging DFS issues.

Usage: `radartool <command> [value]`

Valid commands and values are as listed in [Table 14-1](#).

**Table 14-1 Radartool Command Line Options**

| Command    | Possible Values | Description   |
|------------|-----------------|---|
| usenol     | 0               | Forces the AP to use the same channel even after RADAR is detected on that channel  |
|            | 1               | Normal operation. AP will switch channel on detection of RADAR  |
| dfsdebug   | 0 – 3           | See <a href="#">Table 14-2</a> .  |
| bangradar  | None            | This is a tool to test how software behaves in the event of a successful RADAR detection. Software modules are notified of a successful RADAR detection event |
| shownol    | None            | Displays the channels in the NOL list   |
| enable     | None            | Enables RADAR detection in software   |
| disable    | None            | Disables RADAR detection in software, even for valid DFS channels   |
| numdetects | None            | Prints statistics on the number of RADAR pulses detected  |
| setnol     | channel #       | Adds the specified channel to the NOL list, even if no RADAR was detected in this channel   |

## 14.2.6 Troubleshooting

The easiest way to troubleshoot is to use `radartool` and set DFS debug level to an appropriate value.

**Table 14-2 DFS Debug Levels**

| Debug Level | Debug Output Description  |
|-------------|---|
| 1           | This will print details of every RADAR pulse reported to software. This information will include RSSI value, width of the pulse and timestamp   |
| 2           | In addition to printing details from debug level 1, it will also print the reason why a filter is rejecting a pulse pattern. When it rejects a pulse pattern, details on whether the RSSI value is below a certain threshold and if the pulse width is outside the acceptable range is printed. Also printed is information on whether the PRI/PRF value is outside the margin and if there were enough number of pulses detected |
| 3           | In addition to printing details from debug level 2, FFT related data like pulse bandwidth information and raw RSSI values are printed.  |

Issues you may encounter and possible means to debug those are listed in [Table](#).

### Debug Issues

| Issue  | Possible Debug Method   |
|--|---|
| No or very few RADAR pulses being reported to the software by hardware             | <p>Set DFS debug level to 1. This will print details of every RADAR pulse reported to software. The information will include RSSI value, width of the pulse, and timestamp. From this you can determine how many pulses are being reported to software. In an environment with RADAR pulses, software should be able to detect at least 50% of the pulses.</p> <p>Another cause for this issue might be the time hardware spends transmitting and receiving actual data. If the channel is too busy with too much data, many pulses may be missed or not seen</p>   |
| Receiving some pulses, which could be rejected for not matching any of the filters | <p>Set DFS debug level to 2. In addition to printing details of every RADAR pulse reported to software (RSSI value, width of the pulse, and timestamp), it will also print why the filter is rejecting a pulse pattern. When it rejects a pulse pattern, details on whether the RSSI value is below a certain threshold and if the pulse width is outside the acceptable range is printed. Also printed is information on whether the PRI/PRF value is outside the margin and if there were enough number of pulses detected</p> <p>In case of a chirping pulse, setting debug level to 2 will also indicate if the pulse passes the criteria for chirping pulse detection.</p> |

## 14.3 Zero wait for DFS channels

When an AP comes up in a DFS channel, it is necessary to wait for 60/600 seconds before transmission of frames can begin in that channel. A mechanism to reduce the wait time to zero whenever possible is introduced. This functionality of zero-wait for DFS channels during CAC is supported on the IPQ8064.ILQ.4.0 SPs with QCA9984 chipsets.

The WDS, WDS Enhanced Independent, Ext-AP Enhanced Independent, QWRAP Enhanced Independent repeater modes are supported with various Channel switch options. Radar injection can be done in primary HT80, secondary HT80, or both. Direct attach and offload modes are supported. Support is not implemented for Federal Communications Commission (FCC) domain.

Before the zero-wait for DFS channels was implemented, a wait was noticed after a channel change. Consider a scenario in which Tx is in channel 60. A radar is detected, which causes Tx in channel 60 to be closed and a random channel, such as channel 100, to be selected. After the channel change, a wait of 60 seconds was observed in channel 100 or the CAC expiry allowed to occur without any radar detected before Tx started in channel 100. This wait period is the duration that is eliminated.

With the zero-wait functionality implemented, for European Telecommunications Standards Institute (ETSI) domains, consider the same scenario as previously discussed. Consider a scenario in which Tx is in channel 60 and pre-CAC is in channel 100. A radar is detected, which causes Tx in channel 60 to be closed and a random channel, such as channel 100, to be selected. After the channel change, the wait in channel 100 is prevented if pre-CAC is completed in channel 100. Tx starts without the wait period. Hardware compatibility is necessary for pre-CAC to be performed in one channel and Tx/Rx in another channel. IPQ4019 and QCA9984 chipsets support this capability for zero-wait in DFS channels for CAC.

All regular mode channel-change (except HT80\_80) should work as before. In 80\_80, secondary 80 can be regular or Agile. If secondary segment is Agile then the Tx/Rx in the first segment should remain unaffected during channel change. Initialization is performed of the pre-CAC-required list. A channel is selected from pre-CAC-required list; pre-CAC (60/600secs) is performed and the channel is moved to Pre-CAC-done list. If radar is observed in the secondary segment during pre-CAC, then move the channel to nonoccupancy list (NOL).

New Channel switch logic while moving to DFS channel after radar detection in the primary segment: If the pre-CAC was already done on the new-random-channel then no CAC is required.

Assume the list of all channels comprises [ {42 HT80}, {58 HT80}, {106 HT80}, {122 HT80}, {138 HT80}, {155 HT80} ]. NOL and pre-CAC done lists are empty, whereas pre-AC required list contains [ {58 HT80}, {106 HT80}, {122 HT80}, {138 HT80} ]. User or ACS channel selection is 60HT80==58HT80

Initial channel selection---Although hardware uses both primary and secondary 80, the beacon declares the channel as HT80 only. A channel (say 106HT80) is chosen from Pre-CAC-required list and CAC is started in 58HT80 (segment1) and pre-CAC in 106HT80 (segment2).

Agile CAC and primary CAC expiry---Choose a channel (say 122 HT80) from Pre-CAC-Required list. Primary remains the same and pre-CAC starts in 122 HT80 (segment2).

Radar detection in secondary segment---Primary remains the same. Secondary marks the channel [122 HT80] as radar and adds it to NOL. Pre-CAC starts in 138 HT80 (segment2).

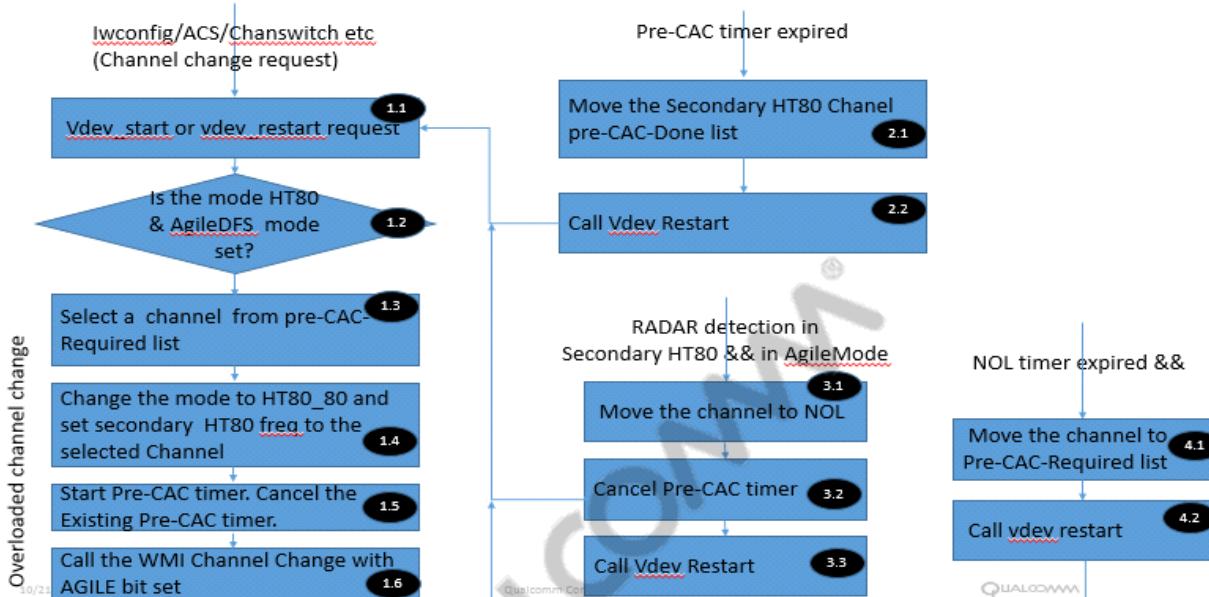
The secondary DFS engine moves to VHT80 or stays in the same channel if it completes the CAC in all the channels. CAC is continued when the Primary DFS engine is moved to NON-DFS channel. When the primary HT80 detects radar, a restart of the secondary DFS engine with the new channel is not necessary.

When the channel is set to 80\_80 and the agile bit is set, it is not guaranteed that firmware does not use 80\_80 mode to send the data packets. The agile Flag for the channel change must not be set when primary HT80 is changing, when the very first channel change (VHT80\_80) occurs after wifi up, and after any primary channel change. This behavior is required because HALPHY bypasses the primary HT80 calibration if it sees the agile flag.

Figure illustrates the workflow of the zero-wait CAC for DFS channels:

## Zero CAC high level flow

22/31



With radar detection in primary HT80 channel, a random channel is selected. A check is made to determine whether the channel is present in the pre-CAC-done list. If the channel is present, the VAPs are brought up and no CAC is required. If the channel is not present, CAC is performed and the VAPs are brought up.

The following are the newly introduced radartool commands:

- Simulate radar in the secondary VHT80; this command is not effective if the secondary channel does not perform pre-CAC (radartool -i wifiX secondarySegmentBangradar)
- Display Precac-required, precac-done and precac-nol lists (radartool -i wifiX showPreCACLists)
- Reset precac-required, precac-done and precac-nol lists (radartool -I wifiX resetPreCACLists)

The following are the newly introduced iwpriv commands:

- Set/Get pre-CAC mode (iwpriv wifiX preCACEn 0/1; iwpriv wifiX get\_preCACEn)
- Set/Get Agile CAC timeout (iwpriv wifiX pCACtimeout 0/1; iwpriv wifiX get\_pCACtimeout)

Set pre-CAC mode and observe the agile lists (Pre-CAC-Required & Pre-CAC-Done) using radartool. Bangradar is present in the Root/Repeater AP during CAC and during Tx.

AgileBangradar is present at the Repeater/RootAP during CAC and during Tx. When channel change and channel switching occur simultaneously, verify zero CAC in the following scenarios:

1. If Pre-CAC is performed on a new channel, then no CAC is observed.
2. When no channel with Pre-CAC is present, a regular CAC is performed.

Throughput drops during secondary HT80 channel change. Changes to connectivity of a mixed set of clients (HT80/20/40/80\_80/160) during secondary HT80 channel change.

## 14.4 ETSI pre-CAC Host driver and firmware enhancements

ETSI pre-CAC is implemented in QCA9984. CAC is normally 60 seconds. For ETSI weather channel, CAC is 600 seconds. During normal channel usage, in-service monitoring is required. CAC results in delay before we can use a DFS channel. The objective is to remove this delay when possible. ETSI allows pre-CAC. In ETSI if CAC is done in a channel, it is valid when either of the following conditions are satisfied:

- Until radar is detected in the channel during normal in-service monitoring
- Until the AP is restarted

ETSI allows Off-channel CAC as long as device meets ISM detection criterion.

- Normal CAC is 60 seconds or 600 sec depending on if we overlap 6600 MHz to 6650 MHz (weather channel)
- Off channel CAC is 6 to 24 minute for normal channel and 1 to 4 hour for weather channel

FCC does not allow pre-CAC or off-channel CAC but there is some room to leverage.

Refer the following documents for more information:

| Title              | Revision | Comment  |
|--------------------|----------|--|
| IEEE 802.11h       |          | IEEE 802.11 standard for Spectrum and Transmit Power Management Extensions |
| IEEE 802.11ac      | 2013     | Very High Throughput WLAN  |
| IEEE 802.11ax      |          | High Efficiency WLAN   |
| IEEE 802.11n       | 2009     | High Throughput WLAN   |
| IEEE 802.11a       | 1999     | High Speed PHY Extensions 5 GHz  |
| IEEE 802.11k, r, y | 2008     | Radio Resource Measurement   |

To understand the zero-wait DFS pre-CAC functionality, become familiar with the following terms:

- Agile Mode or pre-CAC Mode
  - Using a radio or a PHY/DFS engine to detect radar only.
- Channel Availability Check (CAC) Time
  - The time a system shall monitor a channel for the presence of RADAR prior to initiating a communications link on that channel. This is also referred to by the acronym CAC.
- Channel Switch Announcement (CSA)
- Dynamic Frequency Selection (DFS)
- Pre-CAC
  - Doing CAC well in advance of using the channel. This is valid only in ETSI and is not allowed in FCC

HT160 and HT80\_80 mode require two DFS engines for the primary and the secondary. HT80, HT40, HT20 modes use one DFS engine. QCA9984 Agile uses PHY in HT160/HT80\_80 mode but Tx/Rx is on primary channel only. Radar detection continues to be active on both primary and secondary during listen operation. Software announces in beacon that AP is VHT80 mode so that Tx/Rx in HT80. In hardware, two channels – Primary and Agile—are present. In the host AP software, there is one 80MHz channel (the primary) for real-world scenarios. The Host DFS can be Primary and Agile.

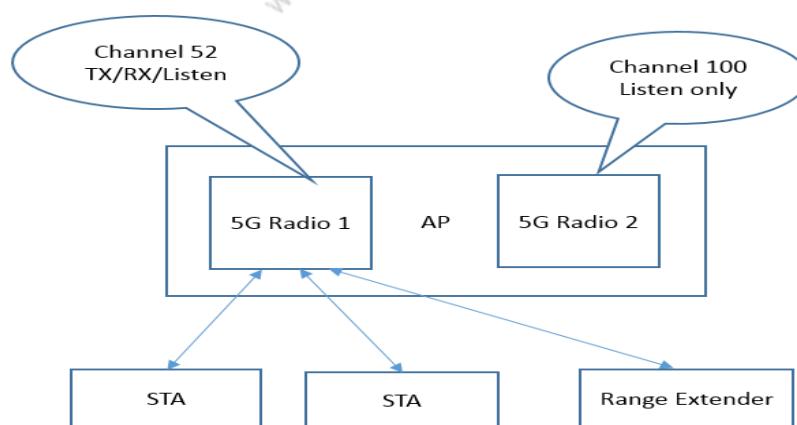
CAC in a DFS channel adds delay and loss of connection. Operation in a DFS channel requires CAC that adds additional delay as well as loss of connection even when CSA is used. ETSI allows pre-CAC, i.e. performing CAC at a convenient time and using that channel later. Once Pre-CAC is done in a channel, it is no longer required to perform a CAC in the channel before TX/RX as long as radar is not found in it or we reset or restart the device.

The zero-wait pre-CAC mechanism applies only to QCA9984 and its specific PHY mode (VHT80\_80, HVT160). It will use only 80MHz primary channel for receive and transmit operation. This is not available in VHT20, VHT40 or VHT80 mode with second PHY/DFS disabled. It is also limited to ETSI domain.

## Theory of operation

**NOTE** The following 2-Radio example is to illustrate the concept of how a second radio can be used to perform pre-CAC. We do not have any plan to implement this in our code at this time.

In a system with two 5G radios, we can use one radio for normal TX, RX, and Radar detection (if the channel is DFS). At the same time the second radio can perform CAC in another DFS channel.



For example the first radio is set to channel 52 to perform CAC followed by transmit/receive/radar monitoring operation and set the second radio to channel 100 to perform CAC.

**NOTE** The first radio is set to a non-DFS channel like 36.

Set the second radio to a DFS channel and listen to the channel for CAC duration. If no radar is detected during that time we can consider that we have successfully finished pre-CAC for the channel. We can switch the channel of the second radio to another DFS channel and repeat the operation. This method is used to prepare a list of channels where CAC is already done (CAC done list).

Before using a channel we can check the CAC done list to see if CAC is required in the channel. We can use a timer task to move from one channel to other for the second radio.

Using a second radio to perform pre-CAC has no consequence for Range Extenders, Clients or WDS modes as this only eliminates the delay introduced by CAC. There is no client attached to the second radio as it does not perform any transmit or receive operation of Wi-Fi frame.

### **Using QCA9984's PHY to perform CAC in a second channel**

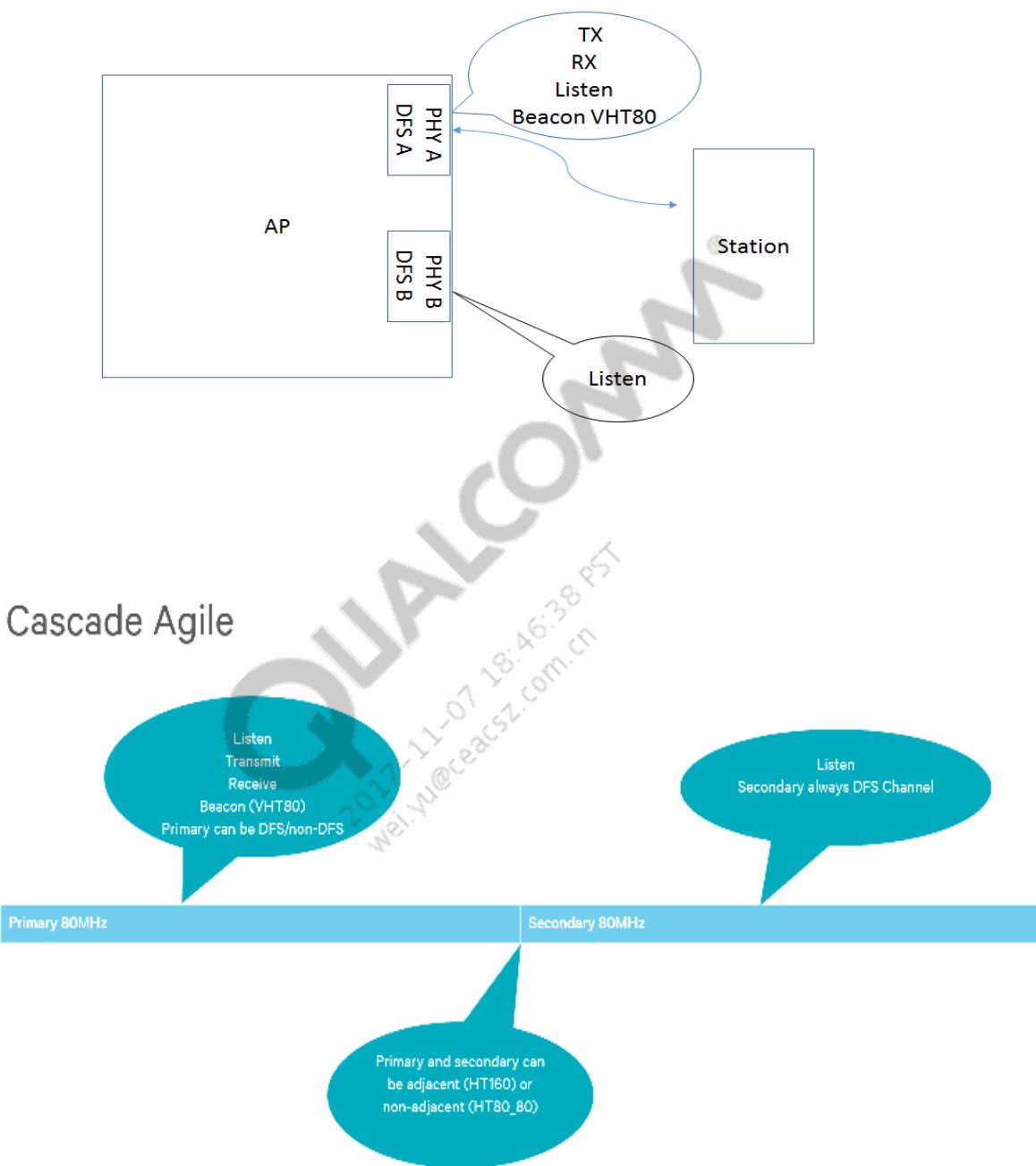
QCA9984 has two 80 MHz wide PHY/DFS engines. Only one of them is used when we operate in VHT80/40/20 mode. Both PHY/DFS engines are used in VHT160 or VHT80\_80 mode.

In VHT160/VHT80\_80 modes we have TX/RX/Listen operation in both 80 MHz channels. However the AP can indicate in the beacon that it is in 80MHz mode instead of real PHY mode of VHT160 or VHT80\_80. Under this circumstance all clients will use only the primary 80MHz channel to connect to the AP. The extension 80 MHz channel will perform listen operation.

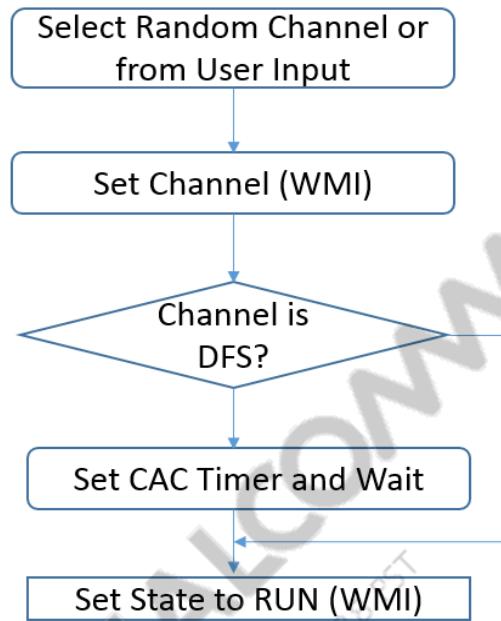
The primary 80 MHz channel uses three chains while the secondary 80 MHz channel uses one chain when there is no transmit or receive operation in progress. Primary 80 MHz channel uses all the four chains when Wi-Fi traffic is present, i.e. for transmission and reception of frames.

The specific mode of QCA9984 described above is termed as QCA9984 Agile mode. The secondary channel is called the Agile Channel.

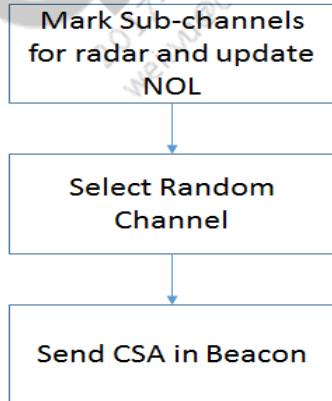
A separate behavior is possible using above hardware configuration, beneficial and allowed for FCC regulatory domains. As above the second radio performs a CAC. But instead of performing CAC's on additional channels, the 2<sup>nd</sup> radio immediately commences in-service-monitoring radar detection on the CAC'ed radar channel. The device may then switch immediately to that backup channel without performing a 60s or 600s CAC. This behavior is allowed for FCC, ETSI and all other regulatory domains. However, this is less beneficial since the second radio is dedicated to maintaining only one backup radar channel.



### Channel Set Sequence without zero-wait pre-CAC



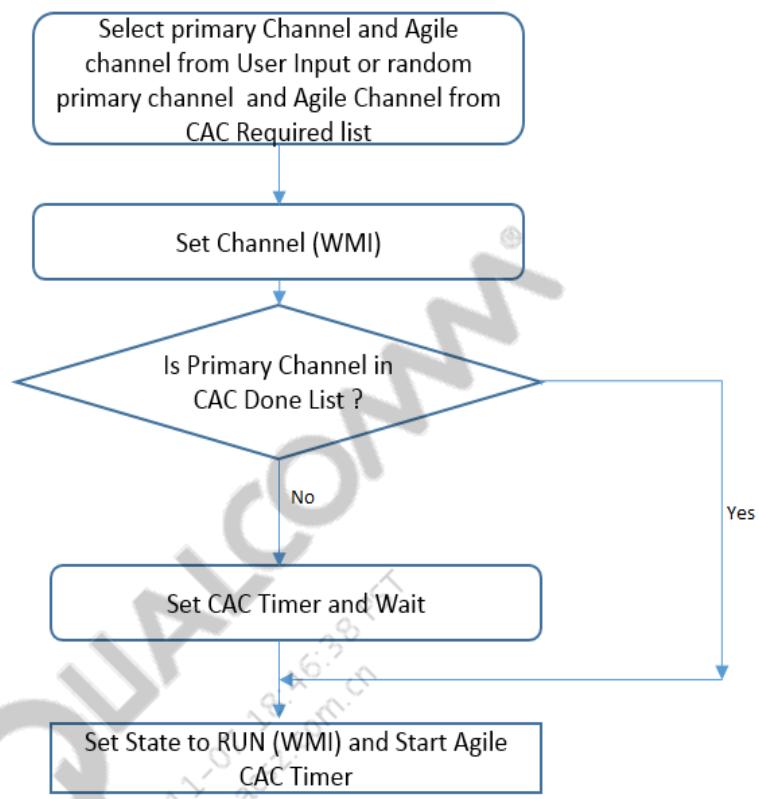
### Radar Detection Sequence without zero-wait pre-CAC



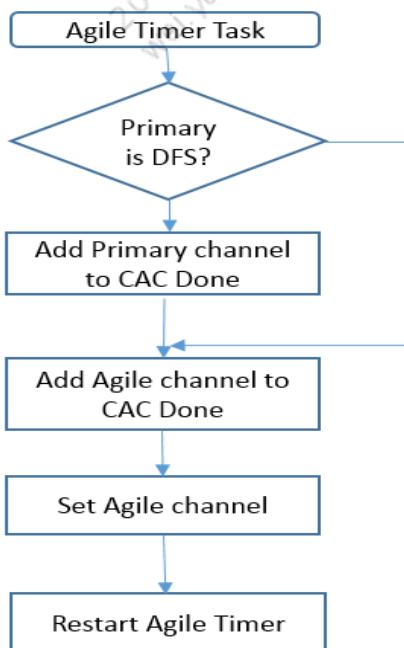
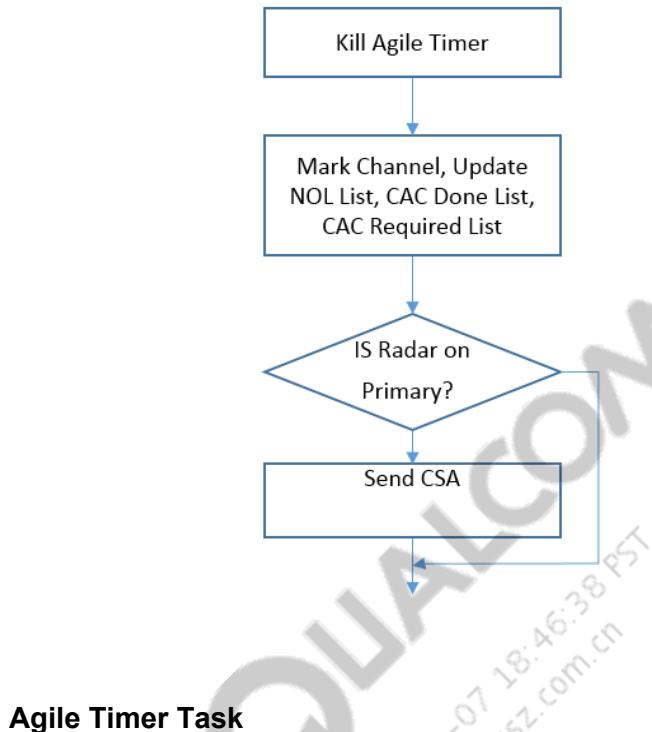
### Agile Timer Task and CAC Done List

- CAC Required List – This is a list of channels for which CAC has not been done. All DFS channels are added to this list at initialization time.
- CAC Done List – This is a list of channels where CAC has been completed. At initialization time this list has no entry.
- Agile Timer Task – This timer provides a mechanism to stay in a channel for required CAC duration and to move on to a new channel at the end of CAC.

## Agile Channel Set Sequence



## Agile Radar Detection Flow



## NOL Timer Task

When the NOL Timer expires, the channel that is freed from NOL list will be added to the CAC Required List.

## UMAC Changes

DFS related changes outlined in previous section require several UMAC changes and enhancements.

- Configuration
  - Method to enable disable the feature through iwpriv and UCI commands.
- Firmware Capability Check
  - Firmware provides information to the host code regarding support of QCA9984 Agile capability.
- Channel Set function Enhancement to indicate presence of 80MHz Agile channel in addition to 80MHZ primary channel.
- Agile channel set function to change to new agile channel when we are done with pre-CAC in current agile channel.
- Beacon changes to indicate 80MHz operation even if the PHY is in VHT80\_80/VHT160 mode in QCA9984 Agile mode of operation.
- State machine changes for Authentication, Association, and Disassociation and so on.

## Firmware Changes

Firmware changes are required for the following:

1. Set channel using primary/secondary/agile info
2. Seamless movement of the agile channel independent of Primary
3. Enabling radar detection on primary and agile channel

## 14.5 EACS PLUS: ACS/OBSS

This section describes the EACS PLUS design. It explains the new ACS metric and the new algorithm as compared to the existing EACS. This section also explains the issues discovered during production testing.

### 14.5.1 Requirements

- EACS PLUS should provide consistent and predictable results.
- Improve the performance of initial EACS vendor testing.
- All the various parameters or metrics used in the ACS should be properly prioritized and provide proper weighting.
- The various metrics used in the ACS should be properly tuned to each scenario.

- EACS PLUS provides all the valid metrics with proper prioritization and valid weighting, while determining the best channel targeting for most scenarios.
- EACS plus provide more run time configurable for customization.
- Support various levels of run time debug option for in-field analysis.
- Analyze each parameter and practical usage. The algorithm need to return to fit to best practical scenario

## 14.5.2 Terminology

| Term | Definition  |
|------|---|
| EACS | Enhanced Automatic Channel Selection                          |
| RSSI | Reduced Signal Strength Indicator                             |
| NF   | Noise floor   |
| OBSS | Other BSS   |
| HW   | Hardware – usually refers to either MAC or CPU on the target. |
| SW   | Software running on the host CPU.                             |
| HAL  | Hardware abstraction  |
| TX   | Transmission  |
| DCS  | Dynamic channel change  |
| CW   | Continuous wave   |
| WLAN | Wireless LAN  |

## 14.5.3 EACS\_PLUS vs. EACS Major Metric Correction

This section describes enhancement done in both the generic algorithm and the ACS metrics used for EACS. The following metric are used for ACS and this section explains the correction done on the metrics to get the best predictable result.

1. avgRssi vs Total Rsssi
2. Noise floor
3. Channel load
4. Adj channel load
5. Regulatory Power
6. DFS channel
7. WEATHER\_RADAR

### 14.5.3.1 Average RSSI vs Total RSSI for each channel for 11NG

The initial 11NG EACS used the average RSSI of all the BSS as the decision making parametric for a single channel. Taking average RSSI prevents a deterministic experimental result. This

provides an invalid result when there are more APs with less signal strength so that the average is greatly reduced compared to a channel having a single AP.

### Example

Channel 1: ap1(65) ap2(2) ap3(1) = avg rssi =  $(65+2+1)/3 = 68/3 = 22.5$

Channel 2: ap1(35) avg rssi =  $35/1 = 35/1 = 35$

As per average RSSI, metric channel 1 is the best channel, but in practice channel 2 is supposed to be the cleanest channel.

EACS PLUS uses Total sum of RSSI, which is 68 for Channel 1 and 35 for Channel 2, so the cleaner channel is Channel 2.

$$\text{CHrssi} = \sum_{k=0}^n \text{BSSrssi}$$

### 14.5.3.2 Max RSSI vs RSSI total for 11NA

The initial 11NA EACS uses max RSSI of an AP in the channel. The most likely scenario is where there are numerous APs with slightly less RSSI, compared to a single AP on a different channel with a little higher RSSI, or all APs in the other channel.

The MAX RSSI will indicate the 1st channel, but actually the 2nd channel is supposed to be cleaner. This is handled by taking Sum RSSI of all beacons in the same channel.

$$\text{CHrssi} = \sum_{k=0}^n \text{BSSrssi}$$

### 14.5.3.3 802.11NG Overlapping Channel (OBSS)

The initial EACS used the average RSSI of all the overlapping channels, which also suffers from the same averaging issue. The issue becomes worse when each channel RSSI is also an average of the all the beacons found in that channel.

### Example

Channel 6 average RSSI, ch1 + ch2 +ch3 +ch4+ /5

Channel 1 average RSSI, ch1 ch2 ch3

So channel 6 will always have a lower average sum as the RSSI is divided by 5.

There are many such practical scenarios where the selection is undesirable. A better scenario would take the weighted average of the overlapping channel, instead of adding to the primary channel for deciding the primary channel

$$\text{PrimCh}_{\text{RSSI}} = \text{CentreChrssi} + \sum_{\text{overlapchannel}} \frac{\text{OverlapChRssi}}{\text{efffactor}}$$

#### 14.5.3.4 Noise floor parameter for 802.11NG

The initial EACS weighted noise floor value is added to the average RSSI metric and the sum is compared between channels. However, noise floor and RSSI are not the same sort of attributes that can be combined together. Noise floor values come in the range of -120 to -80, but RSSI can vary from 0-60; sometimes the average RSSI had much less granularity.

Initial EACS, Summed RSSI = RSSI + weighted \* NF

NF values ranges from -120 when the weight is 2. For 2\*-112, the value goes to -224. But RSSI stays in the range of 60, so most likely RSSI values are ignored. This is more vulnerable when average RSSI is selected as the parameter.

EACS PLUS: NF values are used for rejection only.

1. If (NF > 11NG threshold), reject channel
2. If (NF secondary > threshold), reject the HT40 mode

#### 14.5.3.5 Channel load for 802.11NG

Initial EACS used channel load = Max of center channel vs sec channel

This avoids the scenario of having to distinguish a better channel in the case when one channel has channel load on both primary and secondary, versus the other channel which has load only on the single primary channel.

```
Test Ch1 = Chload on Primary 30, Chload on sec 20
Test Ch2 = Chload on Primary 35
```

The best channel should have been Ch2 but as per Max Channel load criteria Ch1 is selected. To overcome this scenario, the channel load metric selected the channel load of the primary, plus the weighted secondary channel load.

1. Primary Channel Load = Center Channel Load + Secondary Channel load / 2
2. If (Secondary NF > Thres), Primary Channel Load = Center Channel Load + 50

#### 14.5.3.6 ADJ chan channel load metric

Existing EACS

```
max_adj_chan_load = MAX(sec_chan_load, sec_chan_40_1, sec_chan_40_2)
```

Taking Max is not a preferable criteria for metric selection. It is possible that the entire band has consistent channel load, which is worse compared to a band which has a little higher channel load but only at the secondary channel.

`Adj_chan_load` = MAX(sum of adj channel load except primary and secondary, `max_chan_load`)

Comparing the max with a sum is also not recommended. It is preferable to take a weighted sum of channel load.

EACS\_PLUS

$$AdjCh_{load} = \sum_{adjchannel} \frac{AdjLoad_k}{efffactor_k}$$

#### 14.5.3.7 ADJ Chan RSSI metric for 802.11NA

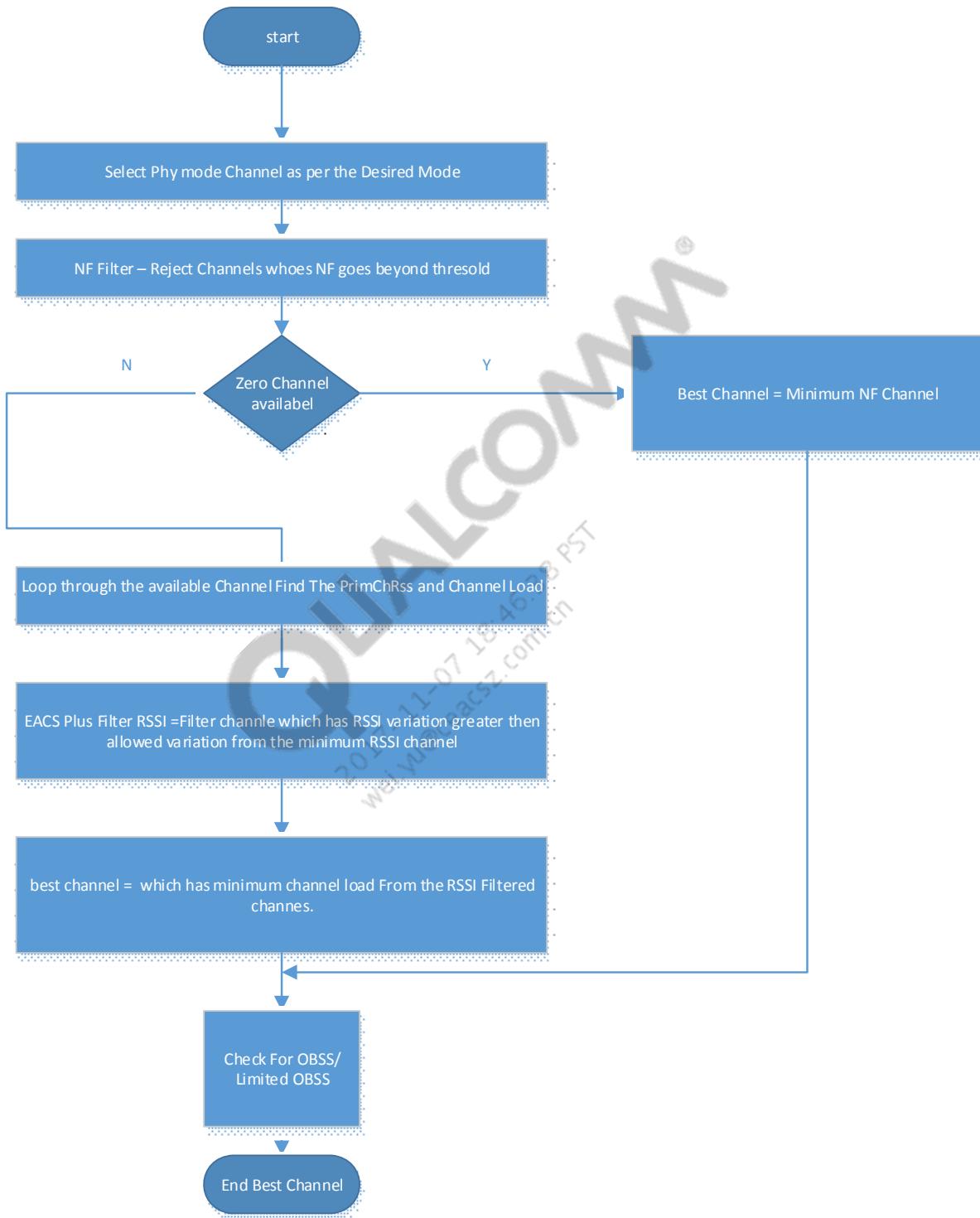
Initial EACS `Adj_chan_rssi` = Max RSSI is seen on all adjacent channels. This is the same issue as explained in [Section 14.5.3.2](#). Max RSSI of channel is not a preferable metric.

EACS\_PLUS

$$AdjCh_{rsssi} = \sum_{adjchannel} \frac{AdjChRssi_k}{efffactor_k}$$

#### 14.5.4 EACS plus algorithm changes

There are many changes done to the original algorithm to fit the new metrics, which were defined and rewritten to satisfy all of the requirements from the initial EACS implementation.

**Figure 14-16 802.11NG EACS PLUS Algorithm**

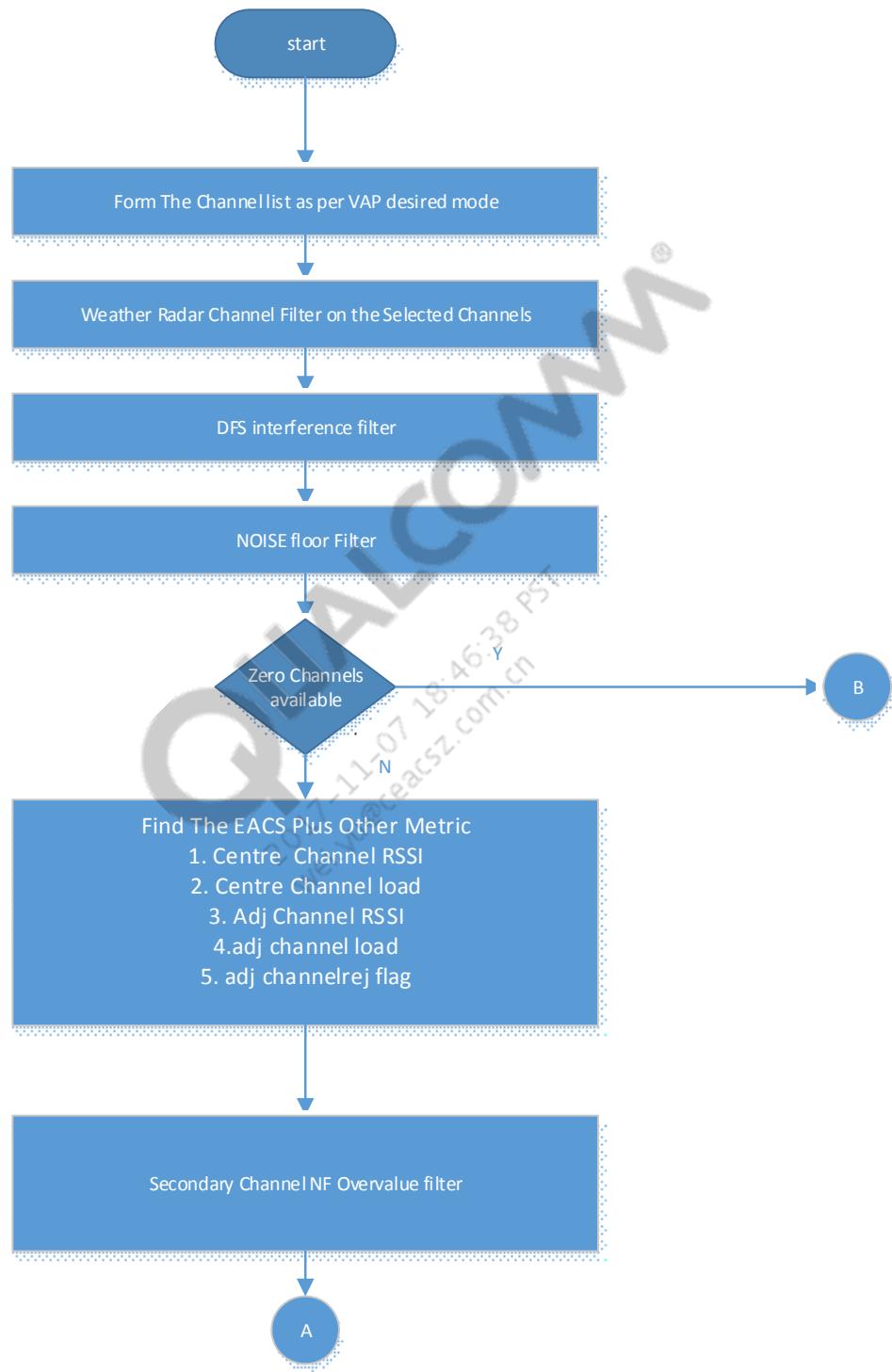


Figure 14-17 802.11NA/AC EACS Plus Algorithm (Part 1)

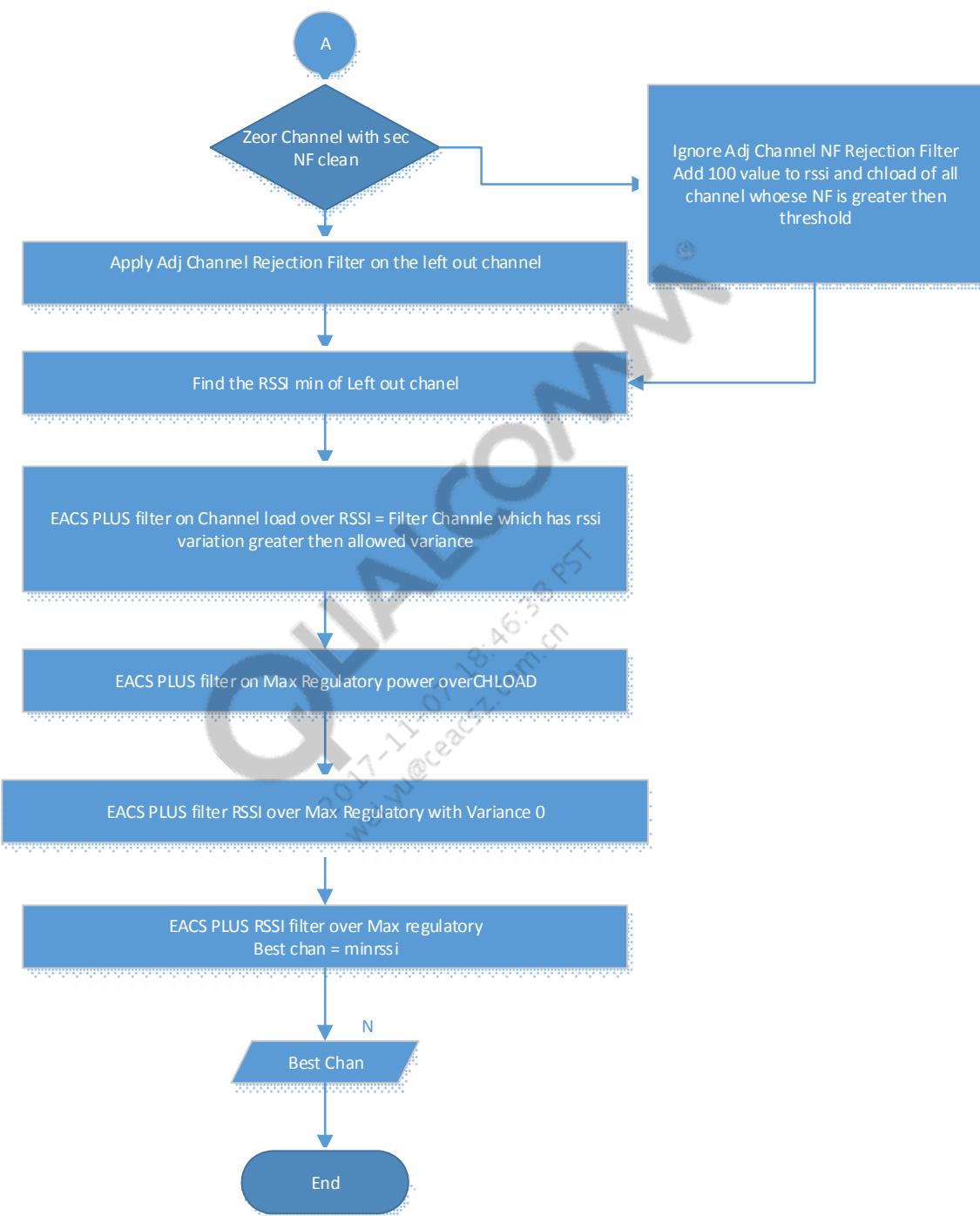


Figure 14-18 EACS Plus Algorithm (Part 2)

### 14.5.5 Back ground ACS/OBSS scan

The primary purpose of this scan timer is to periodically scan the channel when no station is connected to the access point and choose the best channel.

This was a mandatory feature, as once a channel is selected and the AP is active, it is highly likely that channel may eventually deteriorate, or a better channel may become available. It is also possible that another BSS may appear, in which case the access point should turn to HT20 mode.

This has two runtime configurable options, depending on the usage.

- Full ACS scan and select the best channel and mode
- Only OBSS check.

Both of these modes are controlled by a CLI command from iwpriv as listed in the CLI section

### 14.5.6 OBSS check

OBSS scan is done both during AP bootup time, and during the background scan as explained in the previous section.

An OBSS check comes in two varieties, as per different vendor requirements, which is also configured during runtime though an iwpriv CLI command as explained in the CLI section.

- Full OBSS scan

In this mode, all overlapping BSSs are scanned for the particular channel, and if there is an active BSS in overlapping channel, the device goes to HT20 mode.

- Limited OBSS scan

In this mode, the existence of other BSS is checked only for the extension channel, not all of the overlapping channels.

**Table 14-3 iwpriv CLI Commands**

| Command          | Format                               | Description  |
|------------------|--------------------------------------|--|
| acs_bkscanen     | iwpriv wifi1 acs_bkscanen <value>    | Bit 1: Enabled ACS/OBSS background scan depending on the value <i>acs_ctrlflags</i> .<br>Bit 0: Disables ACS/OBSS background scan timer.   |
| get_acs_bkscanen | iwpriv wifi1 get_acs_bkscanen        | ACS/OBSS background scan value   |
| acs_bkscanintvl  | iwpriv wifi1 acs_bkscanintvl <value> | Set the background scan value. Default is one minute.  |
| get_acsscanintvl | iwpriv wifi1 get_acsscanintvl        | Display the background scan timer value  |
| acs_rssivar      | iwpriv wifi1 acs_rssivar <value>     | Set the RSSI variance. Used for ignoring the difference between two channels.<br><br>If the two channels differ with a value less than <i>rssivar</i> , then both channels are considered to have the same RSSI<br><br>Default Value: 30 |
| get_acs_rssivar  | get_acs_rssivar                      | Displays RSSI variance value   |

**Table 14-3 iwpri CLI Commands**

|                   |                                       |  |
|-------------------|---------------------------------------|--|
| acs_chloadvar     | iwpri wifi1 acs_chloadvar <value>     | Set the channel load variance.<br>If two channels differ with channel load value less than ch load variance, they are treated as having same channel load for next level evaluation<br>Default Value: 20   |
| acs_lmtobss       | iwpri wifi1 acs_lmtobss 1             | Enable limited BSS check.  |
| get_acslmtobss    | iwpri wifi1 get_acslmtobss <value>    | Status of limited BSS check enable/disable   |
| acs_ctrlflags     | iwpri wifi1 acs_ctrlflags 0xx         | Background scan ACS control flags<br>0x1 – Full ACS check<br>0x2 -Only OBSS check this is used for manual configuration of channel.  |
| get_acs_ctrlflags | iwpri wifi1 get_acs_ctrlflags <value> | Get value of ACS control flag set  |
| acs_dbgtrace      | iwpri wifi1 acs_dbgtrace 0xxx         | Set ACS run time debug option<br>The values signify the following:<br>EACS_DBG_DEFAULT 0x1<br>EACS_DBG_FUNC 0x8000<br>EACS_DBG_CHLOAD 0x4<br>EACS_DBG_RSSI 0x80<br>EACS_DBG_OBSS 0x100<br>EACS_DBG_REGPOWER 0x200<br>EACS_DBG_NF 0x400<br>EACS_DBG_SCAN 0x800<br>EACS_DBG_ADJCH 0x1000 |
| Get_acs_dbgtrace  | iwpri wifi1 get_acs_dbgtrace          | Display the debug option specified   |

## 14.6 Intelligent Channel Manager

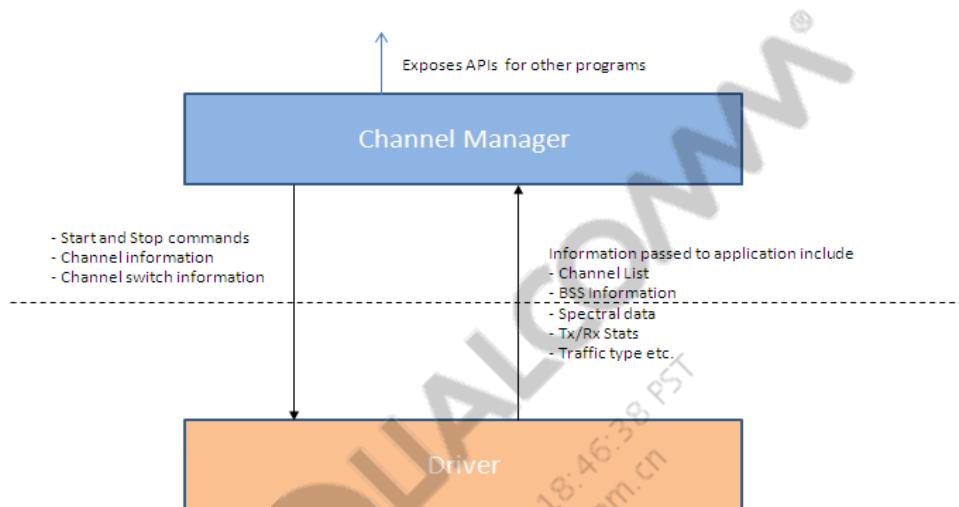
Intelligent Channel Manager (ICM) is an advanced feature for channel resource management available in the AP driver version 10.2 onwards. It primarily provides channel selection and interference mitigation functionality. Currently, some of the channel selection and management features are present as part of the driver framework. ICM is targeted towards moving these features to user space thus introducing flexibility and extensibility for new requirements that may arise in the future. In addition to information from 802.11 scans, ICM also uses Spectral Scans and classification of interference, thus improving decisions. The driver code can view only data for its own radio, while ICM is able to view system-wide data across radios. When third radio sniffer functionality is added, this can be integrated. Historical channel environment data spanning time and locations can potentially be recorded and used, but has not been implemented as of the current release. ICM provides possibilities for much better decision making.

This section describes ICM design and channel selection logic.

## 14.6.1 Design

ICM is designed as a user space application that interacts with the QCA wireless driver. The ICM application interacts with the wireless driver by means of IOCTLs and netlink socket interface. It uses IOCTL interface to set/get commands and a netlink socket to receive asynchronous spectral data and messages from the driver.

A simple block diagram description of ICM is given in [Figure 14-19](#).

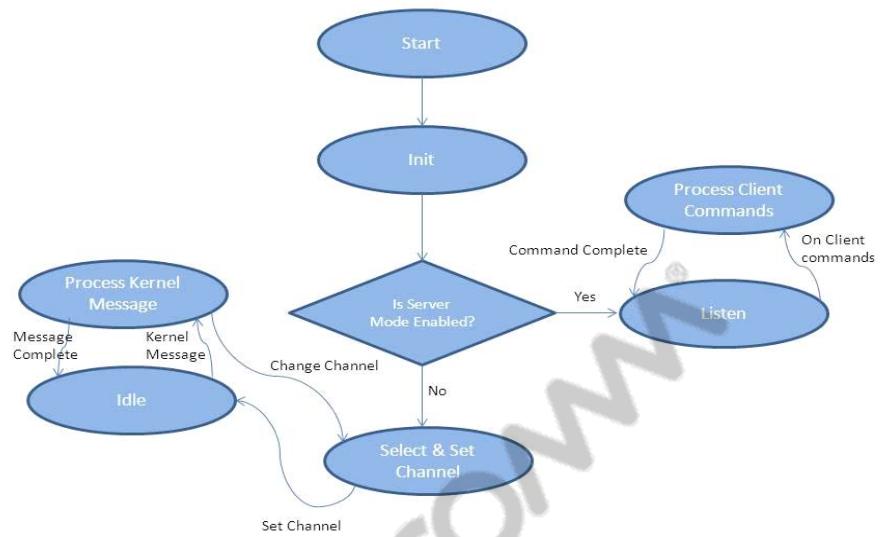


**Figure 14-19 ICM block diagram**

ICM is designed as an application, but for its functionality it depends on the wireless driver. Apart from the standard IOCTL interfaces, a few new IOCTLs are added to get relevant information which is not readily available. These dependencies on IOCTLs are listed in later sections.

### 14.6.1.1 ICM State Machine

[Figure 14-20](#) explains the current state machine for ICM in brief. The two blocks *process client commands* and *process kernel messages* will have their own state transitions.



**Figure 14-20 ICM State Machine**

ICM supports two modes of operation. It can either function standalone or can depend on an external higher layer entity for commands:

- If ICM acts as a stand-alone application, it selects the home channel at VAP bring-up time and will wait for messages from the driver for interference detection and change channel event.
- If ICM works for an external entity, it will not carry out the channel selection; it will function as requested by the external entity.

The main focus is on standalone functionality in this document since interfacing with external entities is currently for QCA internal purposes.

## Init

On the Init state, ICM automatically detects each of the WLAN devices present and initializes necessary data structures for each of them. In addition, ICM opens

- a netlink socket to get Spectral data from the WLAN driver.
- a netlink socket to get interface change notifications from the Kernel Networking subsystem.
- a pipe file descriptor to get internal events from other threads.
- and a UDP/TCP socket to interface with the external entity.

ICM also registers necessary signal handlers to act on exceptions.

## Listen

If ICM is run in server mode, it expects the necessary commands from the external entity. It will not do anything on its own; the functionality of ICM is dictated by the external agent.

## Process Client Commands

When operating in server mode, the commands from the external agent are processed and necessary actions are performed.

### Select and Set channel

When operating stand alone, depending on the user settings, the ICM will select the best channel to operate on. Once the channel is selected and set, it will wait for kernel messages.

### Process Kernel Messages

The wireless driver constantly looks for interference and if detected, will notify ICM via messages. ICM will act on these messages and take necessary actions like changing the channel.

#### 14.6.1.2 ICM IOCTLs

ICM uses the following IOCTLs to configure the wireless driver.

- All Spectral IOCTLs
  - Start Spectral Scan
  - Stop Spectral Scan
  - Get Channel Properties
  - Clear Channel Properties
  - Get Channel Loading
  - Get Nominal Noisefloor value
  - Get Spectral Capability Information
  - Set ICM active
- 802.11 Scan IOCTL
- IOCTL to enable/disable sending of scan channel events by firmware to get NF information (802.11ac chipsets only)
- IOCTL to enable/disable sending of additional IEs (such as HTOP and VHTOP) during 802.11 scan
- IOCTL to set/clear indication in driver that external channel selection is in progress. The indication is set just before start of channel selection, and cleared after setting of channel.
- Channel list IOCTL
- Get Regulatory Domain

#### 14.6.2 ICM Channel Selection Logic

The ICM channel selection logic tries to select the best channel for Wi-Fi traffic, based on the Bandwidth required (160/80+80, 80, 40 or 20 MHz), band (2.4 or 5 GHz) and with inputs from higher layer logic (if present).

### 14.6.2.1 Parameters in channel selection

The parameters used for channel selection are:

1. Number of APs in the current and adjacent/overlapping channels. More the APs, the worse, and lesser the bandwidth available.
2. Interference such as Continuous wave (Video Bridges), which are to be avoided at all cost.
3. Interference such as Microwave ovens which cause packet errors should be avoided.
4. Frequency hopping interference such as cordless phones and Bluetooth can to a certain extent co-exist with WLAN, but should be avoided if possible.
5. As per the 802.11 standard, do not set up an HT40 MHz BSS with the primary on another BSSs secondary 20 MHz, or the secondary on another BSSs primary 20 MHz. There are further details, and they can be obtained from the section on “Channel selection methods for 20/40 MHz operation” in the IEEE 802.11-2012 standard.
6. As per the 802.11ac standard, if an AP selects a primary channel for a new VHT BSS with a 40 MHz, 80 MHz, 160 MHz or 80+80 MHz operating channel width from among the channels on which no beacons are detected during the OBSS scans, then the selected primary channel meets the following conditions:
  - a. It shall not be identical to the secondary 20 MHz channel of any existing BSSs with a 40 MHz, 80 MHz, 160 MHz or 80+80 MHz operating channel width.
  - b. It should not overlap with the secondary 40 MHz channel of any existing BSSs with a 160 MHz or 80+80 MHz operating channel width.

Additionally, An AP should not start a VHT BSS with a 20 MHz operating channel width on a channel that is the secondary 20 MHz channel of any existing BSSs with a 40 MHz, 80 MHz, 160 MHz or 80+80 MHz operating channel width, or is overlapped with the secondary 40 MHz channel of any existing BSSs with a 160 MHz or 80+80 MHz operating channel width.

### 14.6.2.2 Channel Selection Logic - scan and information collection

The scan steps are as follows:

1. Scan all the channels for about 200-300 ms and collect the data regarding the APs present by listening to the beacons. Also, collect the noise floor value in each channel. This is used for purposes of checking whether CW is present (as indicated by elevation of current Noise Floor by a threshold above the nominal). In 2.4 GHz, this serves as a backup for CW interference detection over and above Spectral classification (see point 3). In 5 GHz, this is the only means of detecting CW interference. The nominal Noise Floor for the given chipset is queried from the driver.
2. Channel free time is computed using RX\_CLEAR.
3. In the case of 2.4 GHz channels, perform a separate spectral scan in channels 1, 6 and 11. The scan collects information regarding interference such as CW, FHSS and Microwave Oven. The spectral scan runs in each channel for about 10s.

### 14.6.2.3 Channel Selection - 2.4 GHz

In case of 2.4 GHz band, the data from both the APs as well as spectral analysis is considered. Only channels 1, 6 and 11 are considered. The steps are as follows:

1. All channels are assumed to be fully free to begin with, and have a usability start value of  $2^{16-1}$ . The following steps are used to compute the usability of the channels.
2. The usability value is divided by the number of APs found in the current and overlapping channels + 1. For example, in case of channel 1, 20 MHz mode, the overlapping channels would be 2, 3 and 4. Suppose there are 2 APs in channel 1, 1 in 2 and no APs in 3 and 4, the start value is divided by 3+1. The logic is to figure out what would be the channel free time available when all the APs are transmitting at their peak.
3. To the modified start value from 2, multiply by 60% if FHSS interference is detected and 40% if Microwave is detected. These percentages are heuristically decided, based on expected destructiveness arising from the typical signatures of these interferences.
4. If CW interference is detected, usability is set to zero.
5. In case it is determined that a BSS's secondary 20 MHz is on our (primary) channel, the usability is set to zero. Note that though this is mandatory only to start a 40 MHz BSS, or even any 20 MHz BSS since it would be very disruptive for users and the OBSS otherwise.
6. Further, in case of desired width of 40 MHz:
  - a. The overlapping channels will increase.
  - b. In case an AP is detected in the extension channel, the usability is set to zero
7. In case the measured free time (From RX\_CLEAR), is less than the computed free time, the measured free time is used. This occurs when a few of the APs are hogging all the bandwidth.
8. At the end of the scan, the channel with the highest usability is selected.

### 14.6.2.4 Channel Selection - 5 GHz

In 5 GHz mode, there are no overlapping channels and all the channels are spaced at least 20 MHz apart. Also, there are a greater number of channels to choose from. There are practically no MWO interferences and relatively few FHSS sources. Besides, carrying out Spectral will increase the scan time significantly. So spectral scan is not run. CW interference can be detected using the noise floor, thus saving scan time.

**NOTE** In case of 80+80 MHz, the below channel selection routine is to be invoked twice, first to select the best primary channel, and then to select the best secondary 80 MHz center channel index.

The channel selection runs as follows:

1. In the case of best secondary 80 MHz center channel index selection, we skip all those channels which are already taken up by the best primary 80 MHz, and which lie in either of the 80 MHz segments just adjacent to the best primary 80 MHz segment.
2. Set the usability of all channels to  $2^{16-1}$
3. Scan all the channels for beacons, noisefloor (CW interference) and channel loading

4. If CW interference is detected, set the usability to zero
5. In case of ETSI regulatory domain and weather channel, set the usability to zero (this domain is used in European countries and has a CAC period of 10minutes, so better not use the same)
6. In case it is determined that a BSSs secondary 20 MHz is on our intended primary channel, or that a 160/80+80 BSSs secondary 40 MHz is on our intended primary channel, the usability is set to zero. Both checks are carried out even to start a non-VHT 20 MHz BSSs since it would be disruptive for both users and the OBSS otherwise.
7. (This step is inapplicable to selection of best secondary 80 MHz center channel index).
8. Further, if our desired width is 40 MHz or higher, and an OBSS with primary on our intended secondary 20 MHz channel is found, the channel usability is set to zero.(This step is inapplicable to selection of best secondary 80 MHz center channel index).
9. Further, if our desired width is 160 MHz, and an OBSS with primary on our intended secondary 40 MHz channel is found, the channel usability is set to zero.(This step is inapplicable to selection of best secondary 80 MHz center channel index).
10. In case the measured free time of the primary 20 MHz channel (From RX\_CLEAR), is less than the computed free time, the measured free time is used. This occurs when a few of the APs are hogging all the bandwidth.  
(Going ahead, the span of frequencies across which RX\_CLEAR is checked may be expanded).Adjust usability value on the basis of presence of OBSSs as given in
11. Rank the channels according to their usability, with the most usable channel first.
12. If the usability is the same, rank them according to the maximum Tx power permitted.

#### 14.6.2.5 Adjustment of usability based on OBSS presence

##### Desired bandwidths below 80 MHz

Divide the usability obtained above, by the sum of APs in current and adjacent channel + 1.

##### Desired bandwidth of 80 MHz

The usability of a given primary channel is computed based on

- potential 20/40 MHz PPDU Tx requirement
- potential 80 MHz PPDU Tx requirement

The 20/40 MHz PPDU Tx requirement is affected by

- Competing 80 (or higher) MHz PPDU Tx
- Competing 20/40 MHz PPDU Tx in only our own 40 MHz block, not those in the adjacent 40 MHz block under our 80 MHz block.
- Transmissions in immediate adjacent channel outside 80 MHz block due to ACI.

The 80 MHz PPDU Tx requirement is affected by

- Competing 80 MHz (or higher) PPDU Tx

- Competing 20/40 MHz PPDU Tx in our own 40 MHz block.
- Competing 20/40 MHz PPDU Tx in the adjacent 40 MHz block under our 80 MHz block.
- Transmissions in both adjacent channels outside 80 MHz block due to ACI.

We compute two separate usabilities for each requirement, and then combine the usabilities in a configurable ratio (currently 1:1). While computing the two usabilities, we take into consideration how each is affected to decide which APs to count.

Thus, the 20/40 MHz usability is computed by dividing the existing usability value by the sum of the following:

- Competing 80 (or higher) MHz APs
- Competing 20/40 MHz APs in only our own 40 MHz block, not those in the adjacent 40 MHz block under our 80 MHz block.
- Competing APs in immediate adjacent channel outside 80 MHz block.

The 80 MHz usability is computed separately by dividing the existing usability value by the sum of the following:

- Competing 80 (or higher) MHz APs
- Competing 20/40 MHz APs in our own 40 MHz block.
- Competing 20/40 MHz APs in the adjacent 40 MHz block under our 80 MHz block.
- Competing APs in both adjacent channels outside 80 MHz block.

The final usability is then combined as indicated above.

### **Desired bandwidth of 160 MHz, in 160 MHz (contiguous) mode**

The usability of a given primary channel is computed based on

- potential 20/40 MHz PPDU Tx requirement
- potential 80 MHz PPDU Tx requirement
- potential 160 MHz PPDU Tx requirement

The 20/40 MHz PPDU Tx requirement is affected by

- Competing 160 MHz PPDU Tx.
- Competing 80 MHz PPDU Tx in the same 80 MHz segment as ours, or 80+80 MHz PPDU Tx with our 80 MHz segment as the secondary 80 MHz.
- Competing 20/40 MHz PPDU Tx in our own 40 MHz block.
- Transmissions in immediate adjacent channel outside 160 MHz block due to ACI, if we are at the edge.

The 80 MHz PPDU Tx requirement is affected by

- Competing 160 MHz PPDU Tx.
- Competing 80 MHz PPDU Tx in the same 80 MHz segment as ours, or 80+80 MHz PPDU Tx with our 80 MHz segment as the secondary 80 MHz.

- Competing 20/40 MHz PPDU Tx in the same 80 MHz segment as ours.
- Transmissions in adjacent channel outside 160 MHz block due to ACI.

The 160 MHz PPDU Tx requirement is affected by

- Competing 160 MHz PPDU Tx.
- Competing 80/80+80 MHz PPDU Tx in either segment.
- Competing 20/40 MHz PPDU Tx in our own primary 40 MHz block, or 20/40 MHz PPDU Tx anywhere in the secondary 80 MHz segment.
- Transmissions in both adjacent channels outside 160 MHz block due to ACI.

We compute three separate usabilities for each requirement, and then combine the usabilities in a ratio (currently 35:35:30 for 20/40, 80 and 160 MHz Tx respectively). While computing the three usabilities, we take into consideration how each is affected to decide which APs to count.

Thus, the 20/40 MHz usability is computed by dividing the existing usability value by the sum of the following:

- Competing 160 MHz APs.
- Competing 80 MHz APs in the same 80 MHz segment as ours, or 80+80 MHz APs with our 80 MHz segment as the secondary 80 MHz.
- Competing 20/40 MHz APs in our own 40 MHz block.
- APs in immediate adjacent channel outside 160 MHz block due to ACI, if we are at the edge.

The 80 MHz usability is computed separately by dividing the existing usability value by the sum of the following:

- Competing 160 MHz APs.
- Competing 80 MHz APs in the same 80 MHz segment as ours, or 80+80 MHz APs with our 80 MHz segment as the secondary 80 MHz.
- Competing 20/40 MHz APs in the same 80 MHz segment as ours.
- APs in adjacent channel outside 160 MHz block due to ACI.

The 160 MHz usability is computed separately by dividing the existing usability value by the sum of the following:

- Competing 160 MHz APs.
- Competing 80/80+80 MHz APs in either segment.
- Competing 20/40 MHz APs in our own primary 40 MHz block, or 20/40 MHz APs anywhere in the secondary 80 MHz segment.
- APs in both adjacent channels outside 160 MHz block due to ACI.

The final usability is then combined as indicated above.

## Desired bandwidth of 160 MHz, in 80+80 MHz (non-contiguous) mode

Potential 80+80 MHz PPDU Tx would be a function of the usabilities of each individual 80 MHz block. We do not consider the various combinations of 80+80, and focus instead on finding individual 80 MHz blocks with maximum usability at the 80 MHz level. The approach of considering combinations is not currently expected to yield significant benefits.

Usability processing is done separately for the candidate primary 80 MHz channel, and the candidate secondary 80 MHz channel.

### Selection for segment 0 (primary 80 MHz)

The 20/40 MHz PPDU Tx requirement is affected by

- Competing 160 MHz PPDU Tx.
- Competing 80 MHz PPDU Tx in the same 80 MHz segment as ours, or 80+80 MHz PPDU Tx with our 80 MHz segment as the secondary 80 MHz.
- Competing 20/40 MHz PPDU Tx in our own 40 MHz block. We treat 20 and 40 MHz under the same granularity.
- Transmissions in immediate adjacent channel outside primary 80 MHz block due to ACI, if we are at the edge.

The 80 MHz PPDU Tx requirement is affected by

- Competing 160 MHz PPDU Tx.
- Competing 80 MHz PPDU Tx in the same 80 MHz segment as ours, or 80+80 MHz PPDU Tx with our 80 MHz segment as the secondary 80 MHz.
- Competing 20/40 MHz PPDU Tx in the same 80 MHz segment as ours.
- Transmissions in adjacent channel outside primary 80 MHz block due to ACI.

We compute two separate usabilities for each requirement, and then combine the usabilities in a ratio (current 50:50). While computing the two usabilities, we take into consideration how each is affected to decide which APs to count.

Thus, the 20/40 MHz usability is computed by dividing the existing usability value by the sum of the following:

- Competing 160 MHz APs.
- Competing 80 MHz APs in the same 80 MHz segment as ours, or 80+80 MHz APs with our 80 MHz segment as the secondary 80 MHz.
- Competing 20/40 MHz APs in our own 40 MHz block.
- APs in immediate adjacent channel outside primary 80 MHz block due to ACI, if we are at the edge.

The 80 MHz usability is computed separately by dividing the existing usability value by the sum of the following:

- Competing 160 MHz APs.

- Competing 80 MHz APs in the same 80 MHz segment as ours, or 80+80 MHz APs with our 80 MHz segment as the secondary 80 MHz.
- Competing 20/40 MHz APs in the same 80 MHz segment as ours.
- APs in adjacent channel outside primary 80 MHz block due to ACI.

The final usability is then combined as indicated above.

### Selection for segment 1 (secondary 80 MHz)

The usability is computed based on potential 80+80 MHz PPDU Tx requirement. On the secondary 80 MHz side, it is affected by:

- Competing 160/80+80 MHz PPDU Tx
- Competing 80 MHz PPDU Tx
- Competing 20/40 MHz PPDU Tx anywhere in our secondary 80 MHz span.
- Transmissions in both adjacent channels outside secondary 80 MHz block due to ACI.

The usability is computed separately by dividing the existing usability value by the sum of the following:

- No. of competing 160/80+80 MHz APs
- No. of competing 80 MHz APs
- No. of competing 20/40 MHz APs anywhere in our secondary 80 MHz span.
- No. of APs in both adjacent channels outside secondary 80 MHz block.

#### 14.6.2.6 Handling of case where no channel is usable

The channel selection logic can be called from higher level external logic or standalone. In case of being invoked from higher logic, the channel selection logic returns -1, indicating no channel is good enough. In case the channel selector is invoked standalone from start script, the following action is taken

1. In case of 80 MHz mode, the scan is redone in 40 MHz (both PLUS and MINUS are examined and the best usability considered).
2. In case of 40 MHz mode, the scan is redone in 20 MHz.
3. In case of 20 MHz, if there is no suitable channel available, the first channel with no CW interference is returned.

#### 14.6.3 Interface with Dynamic Channel Selection

In case ICM is running in the background, when the DCS channel change is triggered, the channel change message is given to ICM. In case of 2.4 GHz, a scan is triggered. In case of 5 GHz mode, the next best channel is selected and a channel change is done. If there are no more channels in the list, a full scan is initiated.

## 14.7 UMAC Auto Channel Selection (Scanning)

The goal of Auto Channel Selection (ACS) is to select the best possible operational channel for the AP in the presence of various types of interference.

### 14.7.1 Source of Interference

In the 2.4 GHz channels, the interference is mainly from other WLAN devices, microwave ovens, and CW interference such as Video Bridge. In the 5 GHz band, the interference is mainly from WLAN, CW interference and radar.

### 14.7.2 Selection of Channels

In 2.4 GHz channels, only 3 channels (1,6 and 11) are evaluated. The evaluation is made using the following data:

1. Scan channels 1, 6 and 11. Collect the spectral RSSI, noise floor and other beacon RSSI from other APs. Scan the remaining channels and also collect the same information. Also, from each channel, collect the channel free statistic based on the ANI data
2. The spectral RSSI includes information from current and adjacent Wi-Fi interference and microwave ovens. The noise floor has information on CW interference
3. Add the beacon RSSI from channels around the channels of interest (1, 6 and 11) to the channels of interest. Add a weighted factor of spectral scan RSSI as well as noisefloor. The best channel is one with minimum sum. This channel is chosen for operation.

In the 5 GHz mode, the following logic is used for channel selection:

1. In each channel, find the channel free time (from looking into the ANI counters) and noise floor
2. Once the data for all the channels is obtained, check for high noise floor. If the noise floor is high (indicating CW interference), discard the channel
3. Scan through each channel for the highest channel free time in the current and adjacent channels. If these values are zero, select these channels, starting from the HPA channels.
4. If not zero, find the channel with the lowest value and if more than one have low value, select the one with higher tx power limit.

In VHT mode, if any other AP uses a secondary channel, the AP at hand should not choose that as its primary channel.

The following steps derive the secondary channels of existing APs:

1. Perform a channel scan at 20 MHz, and obtain the PHY mode of the existing APs from the scan results.
2. Based on these PHY modes, derive the secondary channels of the existing APs.
3. Remove the resulting secondary channels from being selected.

If 40 MHz is chosen for the 2.4 GHz band, or 40/80/160 MHz is chosen for the 5 GHz band, the band to be evaluated is accordingly selected (since base bandwidth is 20 MHz), making sure that

the higher and lower channels are correctly selected for evaluation. For 80+80 MHz, the primary 80 MHz channel is chosen as 80MHz channel, then channel selection algorithm is run second time to find the secondary 80 MHz channel. The minimum spacing between primary and secondary 80MHz channel is 80 MHz. When either 2.4 or 5 GHz band may be chosen, the two algorithms are run in sequence. Once the best channel in each band is determined, the channel with maximum free time is selected.

### 14.7.3 Configuration

The ACS feature does not have a specific configuration per se. To trigger the ACS, the AP\_PRIMARY\_CH has to be set to 0. Based on the AP\_CHMODE setting, the 5 or 2.4 GHz selection algorithm gets triggered.

### 14.7.4 Dynamic Channel Selection (DCS)

DCS is a feature to detect and avoid CW interference. CW interference causes the noise floor to be high. This stops transmissions as well as causes receives to fail frequently. The noise floor is monitored by the calibration logic. When the noise floor is above a threshold, the AP is put into EACS mode. It will disconnect from the STAs (it would already have due to the interference) and move to a new channel. The STAs are expected to re-associate with the AP on their own.

In the DCS configuration, the DCS channel switch can be enabled/disabled using the iwpriv command:

```
iwpriv <wifix> dcs_enable <1/0>
```

1 enables DCS and 0 disables the feature. The default value is set to 1.

### 14.7.5 Channel Loading and Channel Hopping

#### 1. Channel Loading

The channel loading feature is primarily an end-user tool to select a channel and provide a histogram of channel loading on each channel. This feature is triggered through an ioctl interface to the upper layer. Channel loading is obtained for each channel netlink interface.

The following is the channel loading requirement.

- Channel loading principle

Due to various interfering radio frequencies and the necessity to provide an optimal quality of service on the physical media at a desired radio frequency, channel loading is used to visualize on request the occupation of WiFi channel(s) (2.4 GHz and 5 GHz).

The proposed solution is divided into two parts: Wireless Site Survey and Wireless Scan

- Wireless Site Survey/Scan

The GUI and ACS must be able to trigger a wireless site survey of the WiFi channels on a per-radio basis. This survey should allow scans of all channels in the identified Wi-Fi spectrum (2.4GHz / 5GHz (11n/ac)) and measure channel occupation/saturation percentages. Occupation/saturation percentages must be calculated channel by channel.

The solution for the Wireless Site Survey must offer the possibility to configure a list of channels to scan. This configuration must be possible for 2.4 GHz and 5 GHz. Duration of the scan to get faster or better measurements.

By default for 2.4 GHz and 5 GHz, scan all channels.

## 2. Channel Hopping

Channel hopping deals with different criteria that should be taken into account before triggering channel change. It is similar to the criteria for DCS and does not significantly interfere in channel selection algorithm. However, it is necessary to maintain history of the blacklisted channel, (channel on which noise was detected) so that the same channel is not used again until a history keeping timer expires (long time default value is 15 minutes).

It is necessary to track noise values for a configurable counter window; another detect window (for polling the register) needs to be started.

### 14.7.5.1 Design Aspects

The following sections describe the basic design aspects for channel loading and channel hopping respectively.

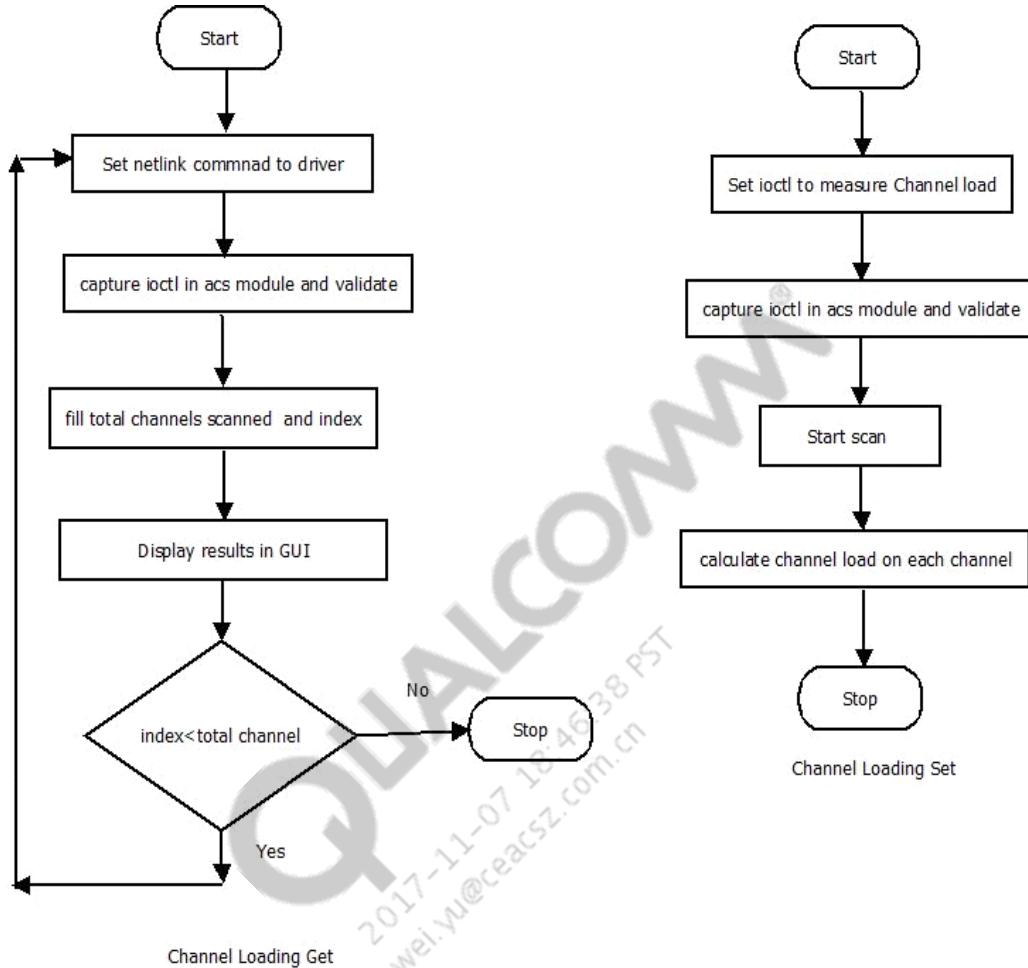
#### Channel Loading

The design allows the user to have complete control of triggering channel loading. The results provided to the user should be the latest.

The feature is part of ACS:

- ACS already has a mechanism to communicate to application layer through netlink sockets and same can be reused for this requirement
- ACS has built-in scan functionality and it can be reused for triggering scan and getting channel load for each channel.

[Figure 14-21](#) shows the flowchart for channel loading functionality.



**Figure 14-21 Channel loading functionality**

### Channel Hopping

The channel hopping mechanism helps create a more stable BSS in term of channel change (with no hop time) and tries to maintain the same channel as much as possible (till the time noise threshold is reached).

The channel is changed as soon as the mechanism detects any CW interference source. The APIs implemented are described in the following sections, and provide control in terms of noise threshold to detect and counter threshold (that is, count the number of times the noise threshold went up in previous measurement interval).

### Architecture Overview

The basic idea of the channel hopping feature is to change the channel based upon noise threshold and counter threshold, which requires the new channel to remain used irrespective of noise level.

If noise level goes high on this channel after the No Hopping time, then the current channel is blacklisted, and the mechanism can skip to next best channel.

While triggering channel change it was found all valid channels got blacklisted. The triggering mechanism is stopped until another bookkeeping timer (long duration) initiates. As soon as the long duration timer begins, the blacklisted channel is enabled for selection in next iteration of counter window (if noise level goes more than threshold).

Figure 14-22 depicts the basic functions of channel hopping.

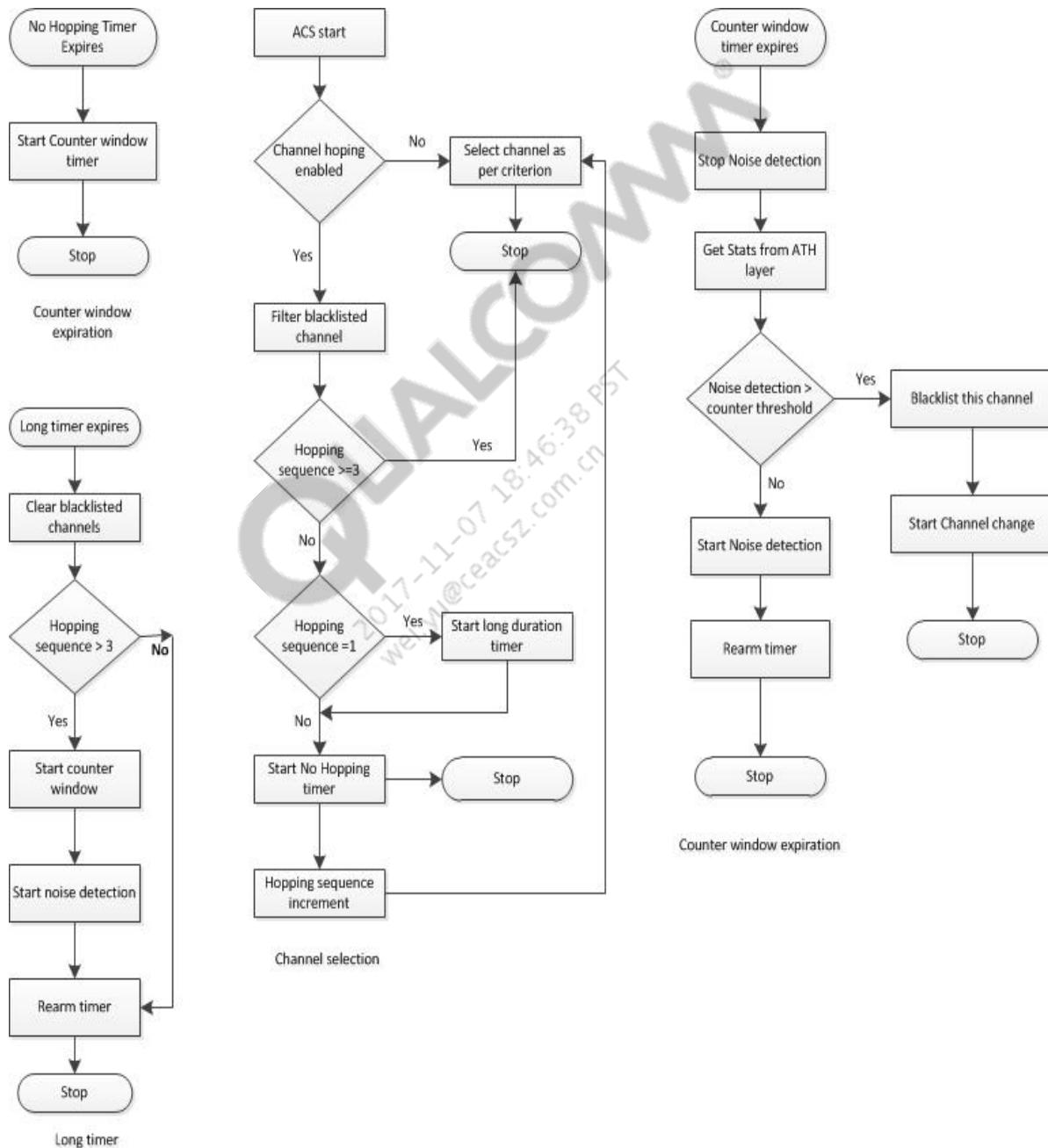


Figure 14-22 Channel Hopping Algorithm

## Functional Description

- At the first instantiation of the channel selection algorithm, channel hopping is enabled. A long duration timer is started, and then a channel hopping window with a default interval of 15 minutes and 1 minute respectively. Simultaneously, noise measurement and counter increment in lower layer (sc) are disabled.
- At expiration of the channel hopping timer, a counter window timer is started and a noise measurement timer in lower layer through the ic API (already defined) is started.
- As ath layer receives noise measurement enabled request, it disables DCS routines and enables a counter in ath\_calibrate routine. There are functions that calculate noise floor, so counters are incremented and this routine gets executed after each 100 ms.
- At expiration of counter window timer the counter percentage which was above then a predefined threshold in last iteration of counter window.
- If this percentage is more than configured percentage and number of hopping are less than three, trigger a channel change. Else, remain on the same channel and do not start the counter window also.
- During counter window channel change operation, blacklist current channel so as to avoid selecting same channel again.
- If actual counter percentage is less than the predefined counter percentage, reschedule the timer and start again.

### 14.7.6 Channel Block List

The ACS feature can be directed to skip one or more channel from being selected by it, by adding such channels to the channel block list. By default, the channels specified are not selected as the primary channel by ACS.

It is possible to set the channel blocking mode to prevent blocked channels from being selected as a secondary channel. ACS works by making a list of channels to evaluate after a scan, and when this feature is used all such blocked channels will be skipped from that list.

It is also possible to set the channel blocking mode to prevent such blocked channels from being selected manually (using iwconfig command). CLI user's guide has details of the channel block list and the channel blocking mode commands.

## 14.8 Non-intrusive channel selection

To support non-intrusive selection of channels, ieee802.11h capability of stations is derived from spectrum management bit of capability information field in association and reassociation requests. Number of non doth supporting station count per radio is maintained in driver. The iwpriv wifiX strict\_doth 1/0 is implemented to enable/disable this behavior. By default, this behavior is disabled.

When channel change is triggered by iwpriv athN doth\_chanswitch <new\_chan> <CSA count> and iwpriv ath0 doth\_ch\_chwidth <new\_chan> <CSA count> <new\_chwidth>, and Iioctl SIOCGATHEXTENDED invocation with subcommand id EXTENDED\_SUBIOCTL\_CHANNEL\_SWITCH, a check is performed to examine whether all the associated stations on all existing vaps support ieee802.11h. If all the existing VAPs support 802.11h, the channel is changed; otherwise, this channel switch request is discarded and the current operating channel continues to be active.

No changes are made to existing DFS behavior and for channel change triggered through iwconfig ath0 channel N. Also, no modifications are performed to channel change triggered by existing DCS implementation (Legacy DCS logic in our WLAN driver). In these scenarios, current default behavior is wlan host driver sends deauth frame to all associated stations and then changes the channel.

In existing driver doth capability is disabled for 2G vaps but still channel switch after channel switch announcement is allowed. To keep backward compatibility, AP will be allowed to send CSA and switch channel irrespective of its own spectrum management capability.

If strict\_doth is enabled, then number of non doth supporting stations will be checked to decide whether AP will go ahead with channel switch or drop this channel switch request. (AP's own channel switch capability will not be checked).

## 14.9 80 MHz Scan

Typical scan uses 20MHz channel width and follow scan logic i.e. listen for beacon in case of passive channel else send probe request for active channel. However, on 11AC architecture, QCA provides a mechanism to use 80Mhz channel width for scan. A new set scan flags for 80MHz scan are introduced.

| Host Flag                | Target Flag                | Description                                    |
|--------------------------|----------------------------|--|
| IEEE80211_SCAN_80MHZ     | None                       | Master flag to enable 80MHz support            |
| IEEE80211_SCAN_PROM_MODE | WMI_SCAN_PROMISCOUS_MODE   | Set Scan with promiscuous mode                 |
| IEEE80211_SCAN_PHY_ERR   | WMI_SCAN_CAPTURE_PHY_ERROR | Allow capture PPDU with phy errors during scan |

Qualcomm Technologies extends the meaning of element of channel\_list array part of wmi\_chan\_list which is passed down to firmware via [WMI\\_START\\_SCAN\\_CMDID](#).

The lower 16 bits (0 to 15) represent the frequency in MHZ and the next 8 bits (16 – 23) represent the WLAN\_PHY\_MODE defined in wlan\_defs.h.

This allows the host to request mode (HT40, VHT40, VHT80) for each channel.



Qualcomm Technologies defines new HTT channel change message [HTT\\_T2H\\_MSG\\_TYPE\\_CHAN\\_CHANGE](#), which is generated by FW every time FW changes channel.

This will be used by host mainly to associate RX frames to correct channel they were received on.

The following field definitions describe the format of the HTT target to host channel change message.

### Header fields

- MSG\_TYPE  
Bits 7:0  
Value: 0xf
- CHAN\_MHZ  
Bits 31:0  
Purpose: Frequency of the primary 20mhz channel.
- BAND\_CENTER\_FREQ1  
Bits 31:0  
Purpose: Centre frequency of the full channel.
- BAND\_CENTER\_FREQ2  
Bits 31:0

Purpose: Centre frequency2 of the channel is only valid for 11acvht 80plus80.

- CHAN\_PHY\_MODE

Bits 31:0

Purpose: phy mode of the channel.

## 14.10 Dynamic Channel Selection-Interference Mitigation (DCS-IM) for 802.11n & 802.11a

The dynamic channel selection algorithm is used to detect the presence of interference and if the interference is detected, switch the network to a cleaner channel.

The DCS function uses a 1 second timer to periodically monitor interference. It checks for the presence of interference, either by directly sensing the channel or in the form of performance degradation. If the interference exceeds thresholds, then it declares that interference is detected. Before changing channel, it waits for some time.

If it detects interference consistently during this period, then it triggers EACS. The Enhanced Auto Channel Scan (EACS) algorithm scans all the channels and chooses a channel that has minimum interference.

This algorithm runs at the AP, so it cannot directly estimate the interference due to a network that is hidden from AP, but it tries to estimate interference based on the current data performance.

This feature is designed for 11NA mode only. 11NG channels are generally very crowded and thus, this feature is not desirable for them.

Since EACS is called for the channel change, traffic gets interrupted when the channel change is triggered. Thus, it may not be desirable to keep DCS always on. Often the user may want to operate on a static channel. Keeping this in mind, DCS is enabled only when the channel is selected by EACS. It does not work when the channel is configured manually.

DCS can be enabled through the command line interface.

### 14.10.1 Implementation

#### 14.10.1.1 Interference Detection Mechanism

##### Parameters Used

The following parameters are maintained by the software and are cleared every 1 second after monitoring the interference.

1. Bytecntrx: counter to count number of data bytes successfully received by the AP.
2. Rxtime: time for which valid packets are received.
3. Bytecnttx: counter to count number of data bytes successfully transmitted by the AP.
4. Wastedtxtime: unsuccessful transmission time

5. Phyerrcnt: counter to count number of PHY errors seen by the software. This will not include the PHY errors filtered by the hardware. Generally, hardware filters OFDM TIMING errors and CCK PHY errors only sends radar errors to the software. This can also be seen as radar pulse error count.

**NOTE** The Tx waste time is not calculated in the QCA9980 firmware as it is CPU-intensive due to software retry.

The following parameters are maintained by the hardware and software reads them from the PHY registers.

1. Cyclecnt: total cycle count from the hardware
2. Rxclrcnt: cycle count for which channel is sensed busy by the hardware.
3. Txfraframecnt: cycle count for which hardware tx\_frame signal is high (indicates time for which hardware is transmitting)
4. Rxframecnt: cycle count for which hardware rx\_frame signal is high (indicates time for which hardware is receiving)
5. Ofdmphyerr\_cnt: OFDM TIMING ERROR count
6. Cckphyerr\_cnt: CCK TIMING ERRORS count

## Description

The DCS function is called every 1 second to detect the presence of interference, except when the current throughput is more than 150 Mbps. When the throughput is already more than 150 Mbps, channel change is not required. This is because 150 Mbps is sufficient for 4 parallel HD Video streams.

DCS is used to detect both co-channel and adjacent channel interference.

## Trigger for co-channel interference

The time for which a channel is used by the co-channel network is unusable for the desired AP. Here, this unusable channel time is used to detect presence of co-channel interference.

Unusable channel time can be computed by measuring the total time for which channel is sensed busy and the time for which channel is used by the desired AP.

Total channel utilization: indicates the time for which channel is busy

$$= (\text{Rx clear cnt diff} / \text{cycle cnt diff}) * 100$$

Tx channel utilization: indicates the time for which AP is transmitting

$$= (\text{Tx frame cnt diff} / \text{cycle cnt diff}) * 100$$

Rx channel utilization: Time for which AP is receiving packets from its own BSS

$$= (\text{Rx time as computed by software} / (\text{OS timestamp diff})) * 100$$

**NOTE** Hardware Rx frame count includes both the activity period of its own BSS and activity period of other co-channel interference. So, here Rx channel utilization

computation is using rx time computed by software instead of hardware rx frame count.

Unusable channel utilization = Total channel utilization - Tx channel utilization - Rx channel utilization

### Trigger condition

When unusable channel utilization is more than the co-channel\_interference\_threshold for at least three seconds in a 10 second window, trigger the channel change.

A counter is maintained to count the number of times interference is detected in a 10 second window. When it exceeds 3, channel change triggers.

Currently co-channel interference threshold is set to 30%. It is programmable and can be changed through a iwpriv command.

### Prevent trigger for far range case

Rx time captures the time for which the AP received packets successfully. It does not include the time for which a packet is received with CRC error. The time for which packets are received with CRC errors will be seen as unusable channel by the algorithm.

As the range increases, more packets will be received with CRC errors. To prevent the false trigger for this case, Rx time is updated for the packets with CRC errors too, when the receive RSSI is lower than a MIN\_RX\_RSSI\_THRESH (by default set to 10 dB).

**NOTE** The extension channel clear count has been changed for QCA9980 and the support for it in the firmware is not added at the time of the initial DCS feature porting, since this requires additional work. This does not impact the detection of co-channel interference and can be implemented as an enhancement.

### Trigger for adjacent channel interference

Packets from adjacent channel network are sometimes seen as energy spikes in the current channel. These packets trigger AGC, but are not detected as WLAN packets by AP. Thus, AP gets OFDM Timing errors whenever it detects such packets in the channel. Because of these timing errors, it is not able to use the channel for successful transmission. Thus, this time can be seen as channel loss due to adjacent channel interference.

Sometimes, packets from the adjacent channel network are observed when there is already a packet in the air by the desired network. In this case, the AP notices an unsuccessful transmission. This time can also be seen as channel loss due to adjacent channel interference.

Based on this behavior, the following two conditions are used to detect adjacent channel interference:

#### Trigger condition 1

Adjacent channel interference causes channel loss due to unwanted phy errors and transmission failures.

Thus, total channel loss in the presence of adjacent channel interference can be computed as follows:

Channel loss due to Phy errors = Phy error count diff \* Phy error penalty

where Phy error diff count is the number of Phy errors observed in 1 sec, and Phy error penalty is the channel time lost due to each phy error. Default value of Phy Error Penalty is set to 500  $\mu$ sec.

Channel loss due to unsuccessful transmission (Wasted tx channel utilization) =  
(Wasted Tx time as computed by software / OS timestamp diff) \*100

Total channel loss (Total wasted channel utilization) =  
Unusable channel utilization + Channel loss due to Phy errors + Channel loss due to unsuccessful transmission

**NOTE** Unusable channel utilization does not include the channel time wasted due to unsuccessful transmission and Phy errors. Total channel loss indicates the total channel time that cannot be used.

When both total channel loss and number of phy errors observed are high, for at least five seconds in a 10 second window and total channel utilization is more than 50%, channel change is triggered. Phy errors are considered to be high when either OFDM timing error count exceeds OFDM\_Timing\_Error\_Thresh or Radar pulse count exceeds Radar\_Phys\_Error\_Thresh. Default value of OFDM\_Timing\_Error\_Thresh and Radar\_Phys\_Error\_Thresh is 300 and 1000 respectively. The thresholds for the two Phy errors are kept different as radar errors may come because of some DFS issues too.

Total channel loss is considered to be high when it exceeds Total\_channel\_loss\_threshold.

Default value of Total\_channel\_loss\_threshold is co-channel\_interference\_threshold (default 30%) + 10%. co-channel\_interference\_threshold can be programmed through command line interface.

## Trigger Condition 2

It is observed that transmission failures due to collision increases in the presence of neighboring channel interference. This results in an overall high transmission failure rate.

When Phy error count is high and transmission failure rate exceeds Tx\_error\_threshold, for at least five seconds in a 10 second window, channel change is triggered.

Where,

Transmission failure rate (Tx\_error) =  
(Wasted tx channel utilization/ Tx channel utilization) \* 100

Default value of Tx\_error\_threshold is 30% and is programmable through iwpriv command. For Phy error count, the same condition as used in above trigger condition is used.

## Prevent trigger for far range case

Wasted Tx time captures the time for which transmission is unsuccessful. As the range increases, the AP operates at a high PER and transmission fails due to poor RSSI. To prevent the channel

change trigger at a far range, wasted Tx time is not updated for unsuccessful packets when Ack RSSI is lower than MIN\_ACK\_RSSI\_THRESH. Default value of this threshold is 10 dB.

#### 14.10.1.2 Criteria

In summary, the following is the criteria to change channel:

1. Co- channel interference is greater than 30%
2. Total channel loss is greater than 40% and Phy error count is high (provided at least 50% of the channel is utilized)
3. Tx failure rate is more than 30% and Phy error count is high.

#### 14.10.1.3 Channel Change Mechanism

When the channel change is triggered, call EACS to find a channel without interference. Switch AP in this channel.

While choosing a clean channel by EACS, ignore all the channels on which interference is already detected by DCS in last 5 minutes.

To avoid false triggers, if channel change is triggered three times in last five minutes, then disable DCS.

### 14.10.2 Configuration

The following command line interface can be used to control DCS algorithm:

- `iwpriv wifi<0/1> dcs_enable <0-3>`  
dcs\_enable <0-3> can be used to disable/enable the DCS feature at run time.  
When dcs\_enable is set to 0, disable DCS.  
When dcs\_enable is set to 1, enable DCS for CW interference mitigation (CW\_IM).  
When dcs\_enable is set to 2, enable DCS for WLAN interference mitigation. Since the algorithm defined in this section primarily mitigates WLAN interferences, we are referring it as DCS for WLAN interference mitigation (WLAN\_IM).  
When dcs\_enable is set to 3, enable both DCS for CW\_IM and DCS for WLAN\_IM.
- `iwpriv wifi<0/1> get_dcs_enable`  
The get function get\_dcs\_enable can be used to get the current status of the DCS.
- `iwpriv wifi<0/1> set_dcs_intrth <value>`  
set\_dcs\_intrth can be used to configure co-channel interference threshold (as a percentage) to trigger channel change. Default value of co-channel interference threshold is 30%.
- `iwpriv wifi<0/1> get_dcs_intrth`  
get\_dcs\_intrth can be used to get the current value of co-channel interference threshold.
- `iwpriv wifi<0/1> set_dcs_errth <value>`

set\_dcs\_errth can be used to configure transmission failure rate threshold, used to indicate presence of interference. Default value of transmission failure rate threshold is 30%.

- `iwpriv wifi<0/1> get_dcs_errth`  
get\_dcs\_errth can be used to get the default value of this transmission failure rate threshold.
- `iwpriv wifi<0/1> set_dcs_phyerrth <value>`  
set\_dcs\_phyerrth can be used to configure channel time wasted due to each Phy error (Phy error Penalty). Default value of Phy error penalty is set as 500  $\mu$ sec.
- `iwpriv wifi<0/1> get_dcs_phyerrth`  
get\_dcs\_phyerrth can be used to get the current value of Phy error penalty (in  $\mu$ sec).

QUALCOMM  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

## 14.11 Modify channel width in CSA for 5 GHz

802.11n channel-width change in channel switch announcement (CSA) is needed. CSA is used before channel changes and when a radar is detected. After CSA is sent, AP changes to another channel or remains in the same channel. However, channel width reduces to ensure that radar channel is not used.

Support is provided to configure dynamic channel bandwidth switching within the operating channel or different channel. This parameter applies for 5 GHz radios only; it is not supported for 2 GHz radios. To switch only the bandwidth and retain the current operating channel, define the same operating channel and enter the new bandwidth. To switch the bandwidth and also switch to a different channel, enter both the new channel and the new bandwidth to be configured.

The ATH\_SUPPORT\_11N\_CHANWIDTH\_SWITCH option is added for this feature. In wlan\_set\_chanswitch(), if current channel is 11N, and channel width change is needed, then a check is performed to determine whether the updated channel width that is specified is valid. If the specified channel width is valid, the channel and channel width are saved, and IEEE80211\_F\_CHANSWITCH flag is set. Then, channel switch announcement is sent after TBTT (time of number of beacons configured), and the channel and channel width are updated accordingly on the AP.

The following is the channel-width change command for the same channel:

```
iwpriv ath0 doth_ch_chwidth <Home Channel> <number of beacons before
change> <new b/w>
iwpriv ath0 doth_ch_chwidth 40 5 20
```

The following is the channel-width change command for a different channel:

```
iwpriv ath0 doth_ch_chwidth <Foreign Channel> <number of beacons before
change> <new b/w>
iwpriv ath0 doth_ch_chwidth 149 5 20
```

## 14.12 Enhanced channel switch and channel scan API

This new feature is built on top of existing scan and channel switch framework and provides more improved control.

Using this new scan, API user space application can provide a set of channel to scan, scan mode, dwell time, normal AP operation time between 2 channel scans from single scan commands.

This new channel scan command provides OBSS utilization, 802.11b presence, Non Wi-Fi interference details (The CW presence) on scanned channel.

New channel switch command enables user space applications to switch the Wi-Fi radio channel from one operating channel/channel width to a new channel/channel width.

Use the iwpriv athN doth\_chanswitch channel tbtt command to force the AP to perform a channel change, and forces a channel change announcement message. Used to test the 802.11h channel switch mechanism. No corresponding get parameter is present.

```
#iwpriv ath0 doth_chanswitch 3 5
```

## 14.12.1 User space application and kernel space driver communication mechanism

### 14.12.1.1 IOCTL based request mechanism

User space application can send any request to kernel WLAN driver using ioctl SIOCGATHEXTENDED.

This ioctl command requires below struct as payload:

Data structure for extended IOCTL command

```
struct extended_ioctl_wrapper {
    u_int32_t cmd;
    void *data;
    u_int32_t data_len;
}
```

| Field    | Description  |
|----------|--|
| cmd      | Sub command request ID.  |
| data     | Data is the payload. Host driver will typecast data pointer to specific type based on cmd. |
| data_len | Size of memory pointed by data field. Typically size of structure                          |

### 14.12.1.2 NETLINK based event notification from WLAN driver

All the notifications from kernel driver to user space application will be based on broadcast netlink events. All events will be sent to netlink socket NETLINK\_ROUTE and group RTNLGRP\_NOTIFY/ RTMGRP\_NOTIFY. Application needs to listen above socket.

## 14.12.2 New IOCTL Commands implemented by WLAN driver

Two new sub IOCTL commands are added to existing IOCTL. One to trigger channel change and other one to scan the channels. These commands are sub commands to SIOCGATHEXTENDED.

The channel scan request can be done by using the existing scan framework for ACS or using the new Wrapper IOCTL call. The existing framework involves to execute more IOCTL to configure the user preferred value. The new IOCTL implementation provides the one short scan request with user preferred scan parameters.

### 14.12.2.1 Using Existing Scan Request framework

- Setting user preferred channel to scan

**IEEE80211\_DBGREQ\_SETACSUSERCHANLIST** command can be used to set the list of user preferred channel for scanning. If use want to set only one channel for scanning then the channel count needs to set to 1 and req.channel\_list[0] to be the desired channel number.

- specify dwell time

- Setting Min Dwell time:

Wlan\_set\_param with IEEE80211\_PARAM\_SCAN\_MIN\_DWELL can be used to set the min dwell time for the scanning.

- Setting Max Dwell time:

Wlan\_set\_param with IEEE80211\_PARAM\_SCAN\_MAX\_DWELL can be used to set the max dwell time for the scanning.

The dwell time will take effective for the passive scan.

- Setting user preferred channel to scan

**IEEE80211\_DBGREQ\_SETACSUSERCHANLIST** command can be used to set the list of user preferred channel for scanning. If use want to set only one channel for scanning then the channel count needs to set to 1 and req.channel\_list[0] to be the desired channel number.

#### 14.12.2.2 New Scan Request framework

The new scan request framework gives the wrapper call to start the scanning with all the user preferred parameters such as user channel list, dwell time, rest time and scan mode setting.

**EXTENDED\_SUBIOCTL\_CHANNEL\_SCAN** sub command is added for triggering scan with below parameters:

Data structure for Channel scan request command

```
typedef struct _wifi_scan_custom_request_cmd {
    u_int32_t          max_dwell_time;
    u_int32_t          min_dwell_time;
    u_int32_t          rest_time;
    bool               scan_mode;
    ieee80211_chanlist_t chanlist
} wifi_scan_custom_request_cmd_t;
```

Data structure for Channel scan request command

```
typedef struct ieee80211_chanlist_r {u_int32_t          max_dwell_time;
    u_int8_t   n_chan;
    u_int8_t   chan[CHAN_SCAN_MAX];
} ieee80211_chanlist_t;
```

Scan request will return 0 in case of success and negative error code in case of failure

| Field          | Description   | Comments   |
|----------------|---|--|
| Max_dwell_time | Max Dwell time on each scan channel in ms                                   | User can configure the Max dwell time for the channel scanning. The expected value should be greater than zero.<br>For the given pool of channel the Max dwell time will keep constant for all the channel scanning.<br>Starting the channel scan will return failure for invalid Max dwell time |
| Min_dwell_time | Min Dwell time on each scan channel in ms                                   | User can configure the Min dwell time for the channel scanning. The expected value should be greater than zero.<br>For the given pool of channel the Min dwell time will keep constant for all the channel scanning.<br>Starting the channel scan will return failure for invalid Min dwell time |
| rest_time      | time in ms between two channel scans for which AP will do normal operation. | Rest_time is defined as the time difference between each channel scanning.<br>The expected value of rest_time should be greater than zero<br>Starting the channel scan will return failure for invalid rest time   |
| Scan_mode      | Scan mode: 0: passive, 1: active  | The scan mode provide two types of scanning option to the user. 1. Active mode 2. Passive mode.<br>By default the channel scan mode will be set as passive mode  |
| n_chan         | Number of channels to scan  | User need to set the number of channel he wants to scan at one request.<br>It is expected that the value always greater than 0 and less than Max channel defined for IEEE channel<br>Starting the channel scan will return failure for invalid number of channel                                 |
| chan           | Array of channel numbers.   | It is user defined list of channel for scanning.<br>Minimum one channel is required to start the scan.<br>The channel should given as per the radio configured channel list<br>Starting the channel scan will return failure for invalid channel   |

#### 14.12.2.3 Scan Response

Existing acs report ioctl **IEEE80211\_DBGREQ\_GETACSREPORT** will be extended to provide all the required info. After issuing scan request, application need to wait for SCAN\_COMPLETE wireless event. After receiving SCAN\_COMPLETE, application should issue acs report IOCTL to get all the channel information. The extended acs report will provide the below additional informations.

- It provides the list of neighbor API details
- Channel OBSS utilization details.
- 802.11b presence in the network
- Non Wi-Fi interference details (The CW presence is notified)

#### 14.12.2.4 Channel change request

EXTENDED\_SUBIOCTL\_CHANNEL\_SWITCH sub command is added for triggering scan with below parameters:

Data structure for channel switch request command

```

typedef enum _wifi_operating_chanwidth
{
    WIFI_OPERATING_CHANWIDTH_20MHZ = 0,                      /* 20 MHz chan width
   */
    WIFI_OPERATING_CHANWIDTH_40MHZ,                            /* 40 MHz chan
width */
    WIFI_OPERATING_CHANWIDTH_80MHZ,                            /* 80 MHz chan
width */
    WIFI_OPERATING_CHANWIDTH_160MHZ,                           /* 160 MHz chan
width */
    WIFI_OPERATING_CHANWIDTH_80_80MHZ,                         /* 80 + 80 MHz chan
width */
    WIFI_OPERATING_CHANWIDTH_MAX_ELEM,                         /* Number of elements in
enum */
} wifi_operating_chanwidth_t;

typedef enum _wifi_sec_chan_offset
{
    WIFI_SEC_CHAN_OFFSET_NA        = 0,                      /* No secondary channel
   */
    WIFI_SEC_CHAN_OFFSET_IS_PLUS = 1,                        /* Secondary channel is
above primary channel */
    WIFI_SEC_CHAN_OFFSET_IS_MINUS = 3,                       /* Secondary channel is below
primary channel */
} wifi_sec_chan_offset_t;

typedef struct _wifi_channel_switch_request
{
    wifi_operating_chanwidth_t target_chanwidth;
    u_int32_t                  target_pchannel;
    u_int32_t                  target_cfreq2;
    wifi_sec_chan_offset_t     sec_chan_offset;
    u_int8_t                   num_csa;
} wifi_channel_switch_request_t;

```

| Field            | Description  |
|------------------|--|
| target_chanwidth | Destination channel width: 0: 20 MHz, 1: 40 MHz, 2: 80 MHz, 3: 160 MHz, 4: 80 + 80 MHz.  |
| target_pchannel  | Destination primary 20 channel center frequency  |
| target_cfreq2    | Centre channel frequency for secondary 80 channel. Used only for 80 + 80 MHz. M must be set 0 otherwise.   |
| sec_chan_offset  | Secondary 20 channel is above or below primary 20 channel  |
| num_csa          | Number of channel switch announcements to send before channel change. If num_csa is specified as 0, default count will be used and as of now its 5. It can be changed any time by changing macro IEEE80211_DEFAULT_CHANSWITCH_COUNT. |

Command will return 0 if channel switch request is accepted else negative error code is returned. If any sanity check fails, command will return negative error code.

- If a channel change request is already in progress, command will return –EBUSY.
- If no active VAP's are present on radio or radio detach is in progress, channel change request will not be honored. In first case error code –EAGAIN will be returned while in second case – EBUSY will be returned.
- AP will be allowed to switch the bandwidth between 20/40/80/160 but it will not be allowed to switch it's mode (Mode switch from VHT to HT or vice versa will not be allowed. similarly 802.11A/B/G/N to 802.11AC or vice versa will not be allowed).
- On receipt of channel change request AP will include channel switch IE along with optional secondary channel offset IE and wide bandwidth channel switch IE in num\_csa beacons.
- Secondary channel offset IE will be included in beacons if AP is operating in NA/NG/AC mode.
- Wide bandwidth channel switch IE will be included as part of channel switch wrapper element in beacons if AP is switching to VHT40 or upper modes.
- Existing DFS handler code will be reused to send channel switch announcement and to change the channel without sending deauth/disassoc to connected stations.
- Many stations operating in 40 MHz channel width don't handle secondary channel switch announcement and assume after channel switch destination 40 MHz channel will be 40PLUS or 40MINUS depending upon their initial association with AP.
- It's suggested that if AP was bring up in a way that it's secondary 20 MHz operating channel is above the primary 20 MHz operating channel (40 PLUS), It should select channels in a way that new secondary 20 MHz channel is again above the primary channel to avoid problems mentioned in above point h.
- It's also suggested that AP should try to maintain existing channel width as far as its possible.
- For 2 GHz band spectrum management capability is not advertised in beacons by default. This was done because some older stations were unable to associate to AP if AP advertises spectrum management capability in 2 GHz band.
- For 5 GHz band spectrum management capability is advertised in beacons by default.

- Spectrum management capability can be enabled or disabled any time by issuing command ‘iwpriv athX vap\_doth 1’ or ‘iwpriv athX vap\_doth 0’ respectively.
- There are some stations which don’t honor channel switch announcement IE’s sent by AP in beacon frames if spectrum management capability bit is not set in capabilities field of beacon.
- Decision to enable spectrum management capability or keep it disabled has to be taken by users judiciously in light of above points.

#### 14.12.2.5 STA stats request / response

To get information regarding number of connected stations, their MAC address, last frames RSSI and so on can be requested by already existing ‘wlancfg athX list sta’ command or IOCTL command IEEE80211\_IOCTL\_STA\_INFO.

Above IOCTL should be called for each VAP (athX interface) to fetch information regarding all the stations associated to this particular VAP. This IOCTL call must provide sufficiently large buffer to WLAN driver and response will be provided in buffer passed in IOCTL request.

Response will carry struct ieee80211req\_sta\_info for each station associated to this particular VAP.

#### 14.12.3 NETLINK broadcast event notification sent by WLAN driver

Driver will send netlink broadcast events to user space

Data structure for netlink based broadcast notification event

```
typedef struct acfg_os_event {
    acfg_ev_id_t id;
    acfg_ev_data_t data;
} acfg_os_event_t;
```

| Field | Description  |
|-------|--|
| Id    | Event ID   |
| Data  | Event data. This is the actual payload this event is carrying. |

For more details please refer driver acfg code base.

##### 14.12.3.1 Channel change notification

Channel change notification will be provided to keep user space application aware of current operating channel. This is particularly required in case WLAN driver changes the channel itself in case of radar detection in current operating channel. acfg framework is already sending this event.

Data structure for channel change notification

```
event ID : WL_EVENT_TYPE_CHAN

Event Data:
typedef struct acfg_chan_start{
    a_uint32_t freq;      /**< Frequency */
```

```

    a_uint32_t          reason;    /**< Reason */
    a_uint32_t          duration;  /**< Duration */
    a_uint32_t          req_id;    /**< Request ID */
} acfg_chan_start_t;

```

### 14.12.3.2 NOL list change notification

A notification will be sent whenever a new channel is added to or removed from NOL list. This notification message will carry number of channels in NOL list and an array containing list of NOL channels. This info can be used by user space application to discard channels in NOL list from scanning and from ranking.acfg framework is already sending this event.

Data structure for channel change notification

```

event ID : WL_EVENT_TYPE_RADAR
Event Data:
typedef struct acfg_radar {
    a_uint8_t          count;
/**< no. of freqs */
    a_uint8_t          freqs[ACFG_MAX_IEEE_CHAN];           /**< list
of radar freqs */
} acfg_radar_t;

```

### 14.12.3.3 BSS Channel utilization periodic notification

WLAN Driver will implement a new periodic event to send self BSS channel utilization, OBSS channel utilization and current noise floor for home channel. This info can be used by user space application to increase or decrease scanning frequency. acfg framework is extended to send this event.

Data structure for channel change notification

```

event ID : WL_EVENT_TYPE_CHAN_STATS
Event Data:
typedef struct acfg_chan_stats {
    a_uint32_t          frequency;

/* Current primary channel centre frequency */

    a_int32_t          noise_floor;

/* Current noise floor on operating channel */

    a_uint32_t          obss_util;
/* Other bss utilization for last interval */

    a_uint32_t          self_bss_util;
/* Self bss utilization for last interval */

} acfg_chan_stats_t;

```

#### NOTE

- Periodic channel stats are sent if obss channel utilization crosses chan\_stats\_th which is by default set as 40 percent.
- Chan stats threshold (chan\_stats\_th) can be read any time by issuing iwpriv command ‘iwpriv wifiX g\_chanstats\_th’ and can be set by issuing command ‘iwpriv wifiX chanstats\_th <NEW\_THRESOLD>’.

#### 14.12.4 Limitations

Scope of this implementation is limited to AP mode only. Ext STA, WDS STA, Repeater mode are being addressed separately.

### 14.13 Inclusion of channel or band capability preference in STA association requests

The methodology to enable a STA to indicate the channel/band capability using Supported Operating Classes element in (Re)Association Request is implemented. With the introduction of this functionality, AP immediately classifies STA as single/dual band and determines whether it supports low 5 GHz, high 5 GHz, or both the bands. AP does not attempt to steer a STA to a channel/band that the STA does not support.

By default, this feature of the STA to provide the channel/band capability is enabled. Both STA and AP must support this IE in the (Re)Association request. The feature is designed to work for dual-band steering only (2 GHz and 5 GHz), and only in environments in which the STA can send their band capabilities in the (Re)Association Request. This mechanism is useful for pre-association time. Currently, the AP cannot classify the STA as lower-5 GHz or upper-5 GHz capable.

The following sequence of operations occur:

1. The STA sends the supporting operating class IE in the (Re)Association request.
2. The WLAN AP driver receives the request and parses it. The driver maps the supported operating class to the regional domain values of the band (2G or 5G) for which it is capable.
3. The WLAN AP driver in the AP sends the band capability (2G/5G/both) of the STA to the LBD daemon running in the AP if band steering feature is enabled.
4. The LBD marks the STA in the database as capable of 2 GHz, 5 GHz, or both and uses it for steering efficiently.
5. Because the LBD knows the STA band capabilities before association itself, steering can be done efficiently.

Consider a sample network in which an AP is brought up with two VAPs (one on each radio with same credentials). Connect a STA that is only 2 GHz-capable to the VAP. The LBD marks the STA as only 2 GHz-capable in the database. The LBD uses this saved information and does not attempt to steer the STA to 5 GHz band.

Later, if a user connects a STA that is both 2 GHz and 5 GHz-capable to the VAP. The LBD marks the STA as both 2 GHz and 5GHz-capable in the database. The LBD then uses this detail and attempts to steer the STA to 5 GHz.

## 14.14 Transmission of QCN IE in beacons, probe responses, and association responses

An access point (AP) transmits QCN\_IE with version info and other attributes in beacon frames, probe responses, and association/reassociation responses. AP will parse probe requests, assoc/reassoc requests and update the node structure with the features supported. Other modules that are dependent on these feature sets take appropriate decision based on the node features.

AP transmits QCN\_IE with QCN Transition Reason Code Attribute in BTM Request Message. AP parses QCN\_IE with QCN Transition Reject Reason Code Attribute in BTM Response Message.

QCN STA advertises the maximum channel time in Probe request FILS Request Parameters IE, which is parsed in the driver and provided to Broadcast Probe request function for appropriate actions.

Valid QCN IE will be at least 8 bytes of size—4 header bytes and 4 version attribute bytes.

**NOTE** A QCN AP may transmit MBO-OCE IE with MBO AP Capability Indication Attribute in addition to QCN IE. This phenomenon indicates that the AP is MBO capable and whether or not it is cellular aware.

If an AP sends MBO Capability Indication Attribute, it does NOT automatically imply that the AP also supports use of MBO Reason codes as defined in SDN 1.0 Proposal pptx. For that we need the AP to transmit QCN IE with QCN Version Attribute with version 1. If an AP only transmits MBO AP Capability Indication Attribute but does not transmit QCN Version Attribute, the behavior of the AP with regards to MBO reason codes is undefined and the QCN STA shall not make any assumptions about it.

If an AP sends QCN Version Attribute with version 1, it DOES automatically imply support for MBO Reason Codes. MBO IE is not additionally required for indicating that.

The same behavior applies with STAs.

### QCN IE Format

| Field          | Size (Octets) | Value (Hex)             | Description   |
|----------------|---------------|-------------------------|---|
| Element ID     | 1             | 0xDD                    | IEEE 802.11 vendor specific information element   |
| Length         | 1             | Variable                | Length of the following fields in the IE in octets. The Length field is a variable, and set to 4 plus the total length of the QCN Attributes. |
| OUI            | 3             | 0x8C-FD-F0              | Qualcomm specific OUI   |
| OUI Type       | 1             | 1 (0 is used by SON IE) | Identifying the type and version of the QCN IE  |
| QCN Attributes | Variable      | Variable                | One or more QCN Attribute(s)  |

## QCN Attribute

| Field                | Size (Octets) | Value (Hex) | Description                                     |
|----------------------|---------------|-------------|---|
| Attribute ID         | 1             | Variable    | Identifies the type of QCN Attribute.           |
| Attribute Length     | 1             | Variable    | Length of the following fields in the attribute |
| Attribute Body Field | Variable      | Variable    | QCN Attribute specific information fields       |

## QCN Version Attribute

| Field                        | Size (Octets) | Value (Hex) | Description   |
|------------------------------|---------------|-------------|---|
| Attribute ID                 | 1             | 0x01        | Identifies the QCN version attribute  |
| Attribute Length             | 1             | 2           | Length of the following fields in the attribute.                              |
| Attribute Body - version     | 1             | Variable    | QCN Version number supported by the device. Current version will be 1         |
| Attribute Body – sub-version | 1             | Variable    | QCN sub-version number supported by the device. Current sub-version will be 0 |

QCN version 1 supports the following AP features:

- AP Steering of Legacy Clients
- Multiple channel steering (root and RE)
- Use of MBO Reason Codes with Unsolicited BTM as defined in SDN 1.0 Proposal pptx
- FILS-SK
- Broadcast Probe Response
- DPP and On-boarding support as defined for QCN1.0

QCN version 1 supports the following STA features:

- Use of MBO Reason Codes with Unsolicited BTM as defined in SDN 1.0 Proposal pptx
- Multiple Channel 11k scan support
- Use of BSS Load Element for roaming (version 1 implementation)
- FILS-SK
- Broadcast Probe Response
- NaN2 usage as defined for QCN 1.0
- DPP and On-boarding support as defined for QCN1.0

## QCN Bit-Mask Attribute

This attribute will be used to identify support for Qualcomm Proprietary features outside of QCN. It will be transmitted only if any such features are supported by the device. As we identify such features, we can update with document with the bit-mask definition. At this point in time, so such features have been identified. Devices receiving this attribute will need to parse it correctly, keep track of features that they also support, and discard any bits that they don't understand.

| Field            | Size (Octets) | Value (Hex) | Description  |
|------------------|---------------|-------------|--|
| Attribute ID     | 1             | Not decided | Identifies the QCN attribute   |
| Attribute Length | 1             | Variable    | Length of the following fields in the attribute.   |
| Feature BitMask  | Variable      | Variable    | Optional Bit-Mask for features supported outside of QCN (e.g. GNSS features, etc.). No features identified at this time and, for now, this attribute will not be transmitted |

### QCN Transition Reason Code Attribute

| Field                  | Size (Octets)  | Value (Hex) | Description   |  |  |
|------------------------|--|-------------|---|--|--|
| Attribute ID           | 1  | 0x06        | Identifies the type of QCN Attribute.   |  |  |
| Attribute Length       | 1  | 0x1         | Length of the following field of the attribute in octets.   |  |  |
| Transition Reason Code | 1  | Variable    | As specified in Transition Reason Codes table, identify the reason for this transition recommendation |  |  |
| Transition Reason Code | <i>Description</i>   |             |   |  |  |
| 0                      | Unspecified  |             |   |  |  |
| 1                      | Excessive frame loss rate  |             |   |  |  |
| 2                      | Excessive delay for current traffic stream   |             |   |  |  |
| 3                      | Insufficient bandwidth for current traffic stream  |             |   |  |  |
| 4                      | Load balancing   |             |   |  |  |
| 5                      | Low RSSI   |             |   |  |  |
| 6                      | Received excessive number of retransmissions   |             |   |  |  |
| 7                      | High interference  |             |   |  |  |
| 8                      | Gray zone<br>Imbalance between the PHY operating margin in the downlink direction compared to the uplink direction can result in a “gray” zone of coverage in which MBO STAs can become “stalled” in certain states such as:<br>Mobile is 802.11 authenticated, but not associated<br>Mobile is 802.11 associated, but not EAP authenticated<br>Mobile is unable to obtain an IP address via DHCP<br>Mobile is unable to perform name resolution via DNS<br>If the serving MBO AP can detect that an MBO STA is in a gray zone, it should try to have the device transition. |             |   |  |  |
| 9                      | Transitioning to a Premium AP  |             |   |  |  |
| 10-255                 | Reserved   |             |   |  |  |

### QCN Transition Reject Reason Code Attribute:-

| Field                            | Size (Octets) | Value (Hex) | Description   |
|----------------------------------|---------------|-------------|---|
| Attribute ID                     | 1             | 0x07        | Identifies the type of MBO Attribute.   |
| Attribute Length                 | 1             | 0x1         | Length of the following field of the attribute in octets.   |
| Transition Rejection Reason Code | 1             | Variable    | As specified in Transition Rejection Reason Codes table, identify the reason that the MBO STA rejected this transition recommendation |

| Transition Rejection Reason Code | Description  |
|----------------------------------|--|
| 0                                | Unspecified  |
| 1                                | Excessive frame loss rate expected by the MBO STA if it transitions to the suggested candidate BSS(s) in the BTM Request frame   |
| 2                                | Excessive delay for current traffic stream would be incurred by BSS transition at this time  |
| 3                                | Insufficient QoS capacity for current traffic stream expected by the MBO STA if it transitions to the suggested candidate BSS(s) in the BTM Request frame  |
| 4                                | Low RSSI in frames being received by the MBO STA from to the suggested candidate BSS(s) in the BTM Request frame   |
| 5                                | High interference expected by the MBO STA if it transitions to the suggested candidate BSS(s) in the BTM Request frame   |
| 6                                | Service Availability – the MBO STA expects that services it needs which are available at its serving AP will not be available if it transitions to the suggested candidate BSS(s) in the BTM Request frame |
| 7-255                            | Reserved   |

In QCN 1.0 timeframe, QCN STA /AP uses QCN IE with Transition Reject Reason code and Transition Reason codes attribute in BTM Response/Request Message respectively, instead of using MBO Transition Reason Code IE and Attribute. Reason for this is to safe-guard against any possible changes to MBO Attribute

## 14.15 Cloud-based channel change

The cloud-based channel change capability is implemented. This technique provides the ability to change device channel triggered from the cloud without interrupting traffic when operating in WDS repeater mode.

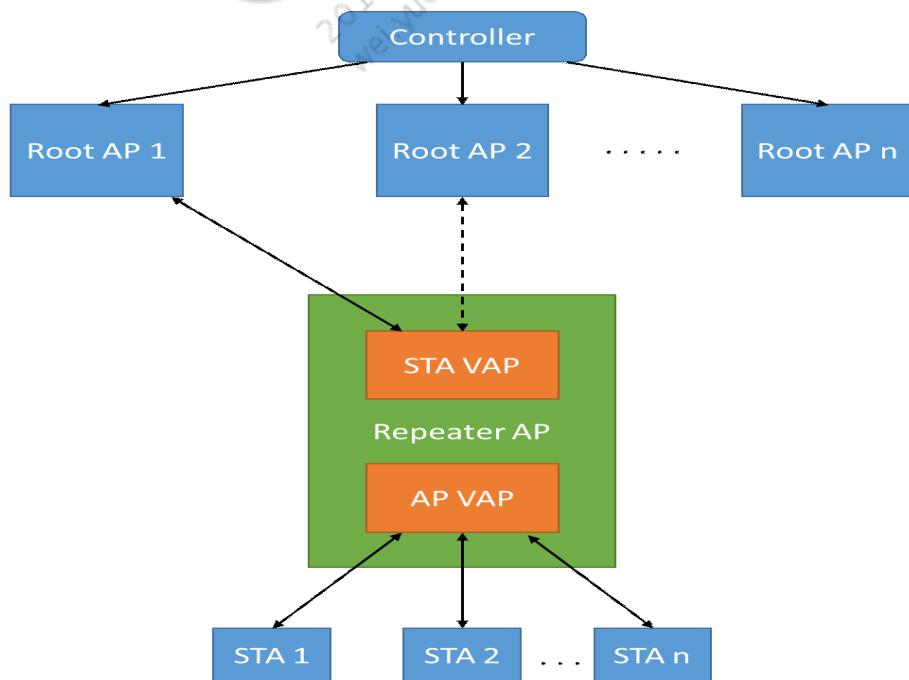
In a configuration where a repeater AP is connected to one of many root APs controlled by a centralized controller, WLAN driver needs to support topology change to a new root AP by seamlessly switching STA VAP from one root AP and associating to another without interrupting any traffic to the clients serviced by the AP VAP.

### 14.15.1 Guidelines for cloud-based channel change functionality

Keep the following points in mind while employing the cloud-based channel change methodology:

- Support for cloud based channel trigger will be implemented for WDS repeater configurations and will not be applicable to Extender AP and QWRAP repeater configurations.
- Cloud triggers channel change by providing at least following information about the new Root AP namely SSID, new Channel number and center frequency of secondary band if applicable. Additional information (if any, E.g. CSA count) may be provided if required.
- Cloud is expected to load the network configurations for all the Root APs that it is controlling into the WPA supplicant configuration file. This can be dynamically achieved through the use of WPA CLI commands. This is made as a requirement in order to aid situations where different Root APs may have different security profiles.
- Cloud will provide bare minimum information as listed above (SSID, channel number etc.) that will be used by the design to perform a home channel scan and update the scan entry for new Root AP.
- Trigger to change to new Root AP assumes DFS checks have already been performed by new Root AP and hence STA will bypass radar detection.
- After topology change, Root AP routing table updates will be taken care by controller (entity that triggers channel/AP switch) and is assumed as out of scope for this FR.
- On receiving trigger from cloud, Channel Switch Announcement (CSA) is issued on the AP VAPs following which STA VAP will associate with new Root AP on the given channel.

The following figure illustrates the scenario for cloud-based channel change:



**Figure 14-23 Scenario for cloud based trigger**

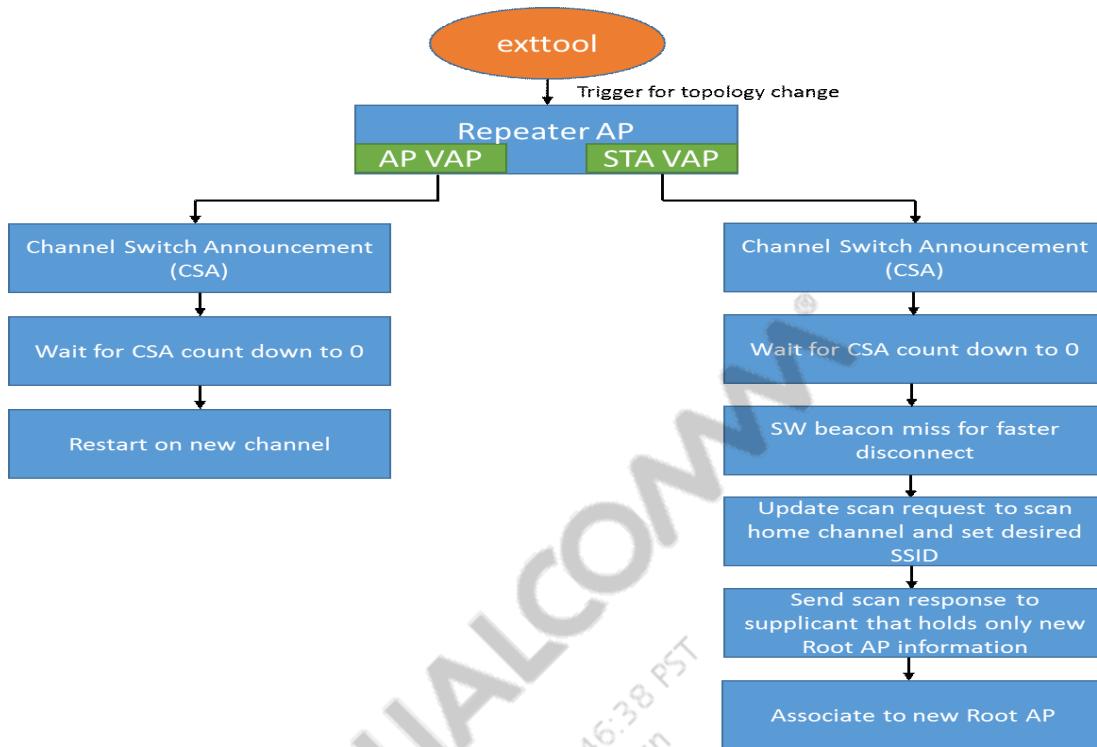
The preceding illustration shows one of the scenarios under which a cloud based channel trigger can occur. In the configuration above, all the Root APs are controlled centrally by a single controller.

Consider a Repeater AP associated to Root AP1 as shown in the preceding figure. Two VAPs namely STA VAP and AP VAP are created in the Repeater AP wherein STA VAP is associated to Root AP1 and AP VAP services clients STA1, STA2 and so on. Note that it is also possible there are multiple AP VAPs created in the Repeater AP each servicing its own set of clients associated to it.

During the cloud trigger to change to new Root AP (say Root AP2 in this case), our driver will issue CSA on the AP VAP(s) to indicate its clients about channel change. Once CSA count reaches 0, AP VAP(s) and its clients will move to the new channel. During the same time, STA VAP will restart on the new channel and associate with new Root AP (Root AP2 in this case) in the given channel.

#### 14.15.2 Flow diagram of repeater AP behavior

The following figure shows the flow the behavior of repeater AP's STA VAP and AP VAP(s) during cloud based trigger to change topology for both the modes of implementation:



**Figure 14-24 Flow diagram describing AP and STA VAP behavior during topology change for Scan mode**

To trigger the topology change, existing exttool application implemented in our WIN host driver will be leveraged with minor modifications to the input parameters as required.

The preceding diagram illustrates the Repeater AP's STA VAP and AP VAP behavior during topology change. Note that only a single AP VAP is depicted for illustration purposes. However, there may be multiple AP VAPs created in the Repeater AP. Behavior remains the same across all the AP VAPs of the Repeater AP during topology change.

### 14.15.3 AP VAP behavior during cloud-based channel change

- Topology change to switch to new Root AP is triggered by leveraging exttool application
- When the topology change lands on our driver, Channel Switch Announcement (CSA) is issued on all the AP VAPs created on the Repeater AP
- CSA count is set to user specified value, or by default set to 5 beacon counts
- After issuing CSA, AP VAP waits for the CSA to count down to 0

- After CSA count reaches 0, AP VAPs restart on new channel provided as part of the topology change
- At any point during the process of STA VAP association with new Root AP, AP VAP is maintained to be in UP state to ensure the clients being serviced do not face disconnect

#### 14.15.4 STA VAP behavior during cloud-based channel change

- Topology change to switch to new Root AP is triggered by leveraging exttool application.
- When the topology change lands on our driver, STA VAP waits for the CSA count issued on AP VAP(s) to reach 0.
- After CSA count reaches 0, STA VAP restarts on the new channel provided as part of topology change and immediately issues a SW beacon miss to achieve faster disconnect with old Root AP.
- On disconnecting with old Root AP, WPA supplicant issues a scan request. STA VAP updates the scan request to scan home channel and update SSID list to hold only that of the new Root AP.
- Scan result received is filtered out to hold only scan entry of the new Root AP. This is sent to the supplicant as scan response.
- Supplicant receives the response and triggers association process with new Root AP.

Throughout the process, STA VAP ensures the AP VAP(s) are not brought down due to connection state change. In other words, functionality of enhanced independent repeater mode is replicated to ensure AP VAP(s) always remain in VAP UP state.

#### 14.15.5 Limitations with cloud-based channel change

The following limitations apply:

- Instead of using a WPA CLI command that will enable to add network to the supplicant configuration and update it, currently, the configurations are updated initially and then the supplicant is restarted for the configuration to take effect.
- In 2G WDS dependent configuration, by default the 802.11h flags are cleared based on the spectrum capability information. Due to this CSA may not go through. Hence, for testing the feature on 2G, the `vap_doth` flags for the VAPS may need to be set using `-iwpriv athX doth 1`.

#### 14.15.6 Sample configuration scenarios

As a prerequisite to all the verification procedures described in this section, it is assumed that all the root APs that must be controlled by the cloud need to be configured with its network profile in the supplicant configuration file. Perform the following steps:

1. Terminate the `wpa_supplicant` using "killall `wpa_supplicant`"
2. Add all network configurations in `var/run/wpa_supplicant-ath0.conf`

3. Restart wpa\_supplicant using "wpa\_supplicant -P/var/run/wifi-ath0.pid -Dathr -iath0 -bbr-lan -c /var/run/wpa\_supplicant-ath0.conf -B"

**To verify whether this cloud-based channel trigger feature is invoked in 5G band:**

Update supplicant config file to hold network profile for new Root AP.

Using usual UCI config, setup a basic WDS repeater configuration using a Root AP and Repeater AP. In repeater AP, execute the following command for STA VAP interface -exttool -i wifiX -p -a <Chan> -q <SSID>. Channel switch announcement will take place and repeater AP will move to new channel.

On new channel, STA VAP will trigger association process with new Root AP having SSID that is provided by user.

**To verify whether this cloud-based channel trigger feature is invoked in 2G band:**

Update supplicant config file to hold network profile for new Root AP. Using usual UCI config, set up a basic WDS repeater configuration using a Root AP and Repeater AP. In repeater AP, execute the following command for STA VAP interface -exttool -i wifiX -p -a <Chan> -q <SSID>. Channel switch announcement will take place and repeater AP will move to new channel.

On new channel, STA VAP will trigger association process with new Root AP having SSID that is provided by user.

**To verify the checks for interface, channel number and SSID information provided by user:**

Update supplicant config file to hold network profile for new Root AP. Using usual UCI config, set up a basic WDS repeater configuration using a Root AP and Repeater AP.

In repeater AP, execute the following command for STA VAP interface using -

1. Invalid Interface
2. Invalid Channel (Not a certified WLAN 2G/5G channel)
3. Invalid SSID (Length > 32) exttool -i wifiX -p -a <Chan> -q <SSID>. On issuing extttool using invalid input parameters, one should observe the application complains about the invalid parameter and returns error. No action will be carried out.

**To verify multiple Root AP network configurations in supplicant in different and same channel:**

Update supplicant config file to hold multiple network profiles for new Root AP. Using usual UCI config set up a basic WDS repeater configuration using Root AP and Repeater AP.

Bring up Repeater AP and ensure scan request is received from supplicant with provided SSIDs in the network configuration. Find out the maximum allowed configurations in the supplicant config. On bringing up the repeater AP, scan request is issued by the supplicant with SSID list containing all the SSIDs that are provided in the network supplicant configuration file.

**To verify repeater move to a new channel:**

Update supplicant config file to hold network profile for new Root AP. Using usual UCI config, setup a basic WDS repeater configuration using a Root AP and Repeater AP. In repeater AP, execute the following command for STA VAP interface -

`exttool -i wifiX -p -a <Chan> -q <SSID>`. Channel switch announcement will take place and repeater AP will move to new channel.

On new channel, STA VAP will trigger association process with new Root AP having SSID that is provided by user.

**To verify repeater move back to the previous Root AP:**

Using usual UCI config, setup a basic WDS repeater configuration using a Root AP and Repeater AP. In repeater AP, execute the following command for STA VAP interface `-exttool -i wifiX -p -a <Chan> -q <SSID>`. Channel switch announcement will take place and repeater AP will move to new channel.

On new channel, STA VAP will trigger association process with previous Root AP having SSID that is provided by user.

**To verify the repeater movement to new Root AP with different security profile:**

Update supplicant config file to hold network profile for new Root AP. Ensure security related information is contained in the config for new Root AP.

Using usual UCI config, setup a basic WDS repeater configuration using a Root AP and Repeater AP.

In repeater AP, execute the following command for STA VAP interface `-exttool -i wifiX -p -a <Chan> -q <SSID>`. Channel switch announcement will take place and repeater AP will move to new channel. On new channel, STA VAP will trigger association process with previous Root AP having SSID that is provided by user. Keys should be exchanged and association process be completed successfully.

**To verify whether the repeater movement takes place without bringing down AP VAPs in repeater AP:**

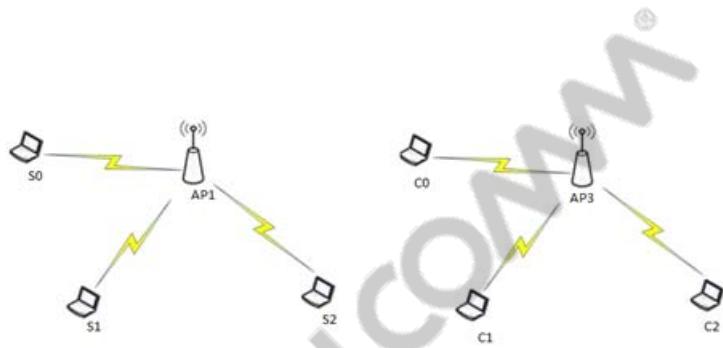
Update supplicant config file to hold network profile for new Root AP. Using usual UCI config, setup a basic WDS repeater configuration using a Root AP and Repeater AP. Ensure to have atleast 1 AP VAP in the repeater AP.

In repeater AP, execute the following command for STA VAP interface `-exttool -i wifiX -p -a <Chan> -q <SSID>`, Channel switch announcement will take place and repeater AP will move to new channel.

On new channel, STA VAP will trigger association process with previous Root AP having SSID that is provided by user. Throughout the process, AP VAP of the Repeater AP will remain in up state.

## 14.16 Scanning RSSI of NACs

This section describes the functionality to obtain the RSSI of nonassociated clients (NACs) after enabling the scanning of RSSI of NACs. Consider a sample scenario as illustrated in the following figure. Clients C0, C1, and C2 are associated with AP3. AP1 needs to obtain the RSSI of client C0. BSSID of AP3, MAC address of C0, and the channel on which AP3 is working are provided on AP1 when AP1 receives the RSSI of C0. Essentially, the RSSI is used for roaming between the two APs. Users can learn about the RSSI of the client to decide the AP with which the client must be associated.



No new dedicated VAP is used for the NAC RSSI obtaining as VAP numbers are limited in certain customer deployments. The BSSID, MAC address of the client and channel are configured using CLI commands. If the configured channel is different from the working home channel on AP, an off-channel scan is performed to sniffer the frames from the client to its associated AP. A wlanconfig CLI command to perform the scanning of RSSI of NACs by capturing the frames between the client and its associated. Also, a wlanconfig command to show the RSSI of the NAC is implemented.

If the channel that is specified as the input parameter is the home channel, the BSSID and MAC address of the client are set in to firmware (FW) to receive the frames from the client to the configured BSSID. The RSSI is recorded in FW. If the channel is not equal to the home channel, off-channel scan is implemented after setting BSSID and MAC address of NAC to FW.

No special VAP is dedicated to perform the scanning of NAC RSSI.

To perform the scanning of RSSI of NACs, enter the following command:

```
wlanconfig athX nac_rssi add/del bssid xx:xx:xx:xx:xx:xx client
xx:xx:xx:xx:xx:xx chan value
```

The MAC addresses of BSSID and client are specified. The channel value can be 0, in which case the current channel will be used.

To display the scanned RSSIs of NACs, enter the following command:

```
wlanconfig athX nac_rssi list
```

When the BSSID, client and channel are added by CLI, host performs the validity check of the parameters, such as valid unicast MAC format and valid channel number, which is the same as the validation performed with other wlanconfig athX commands.

If the parameters are valid, BSSID is set to register RXPCU\_BSSID3\_L32, RXPCU\_BSSID3\_U16 and RXPCU\_RX\_FILTER2, PROXY\_STA\_AD1\_SEARCH\_BSSID, and the client MAC is stored in FW. If the configured channel is different from the working channel, switch the channel to the configured channel and then come back in 300 ms (refer to the exttool --scan --interface %s%d --mindwell %d --maxdwell %d --resttime %d --scanmode 1 --chcount %d %s command).

The wlanconfig athX rss1\_nac add command can be entered multiple times with same BSSID and client MAC continuously. When this command is entered the first time, BSSID and client MAC are set in the FW. For all the subsequent entries of this command, only a change in channel is performed, if needed. However, if the BSSID or client MAC are different from the previous settings, an error is returned.

If the BSSID and client MAC are not specified when del parameter is set with the wlanconfig command, an error message is displayed.

The command list contains the BSSID, MAC address of the client, and the average RSSI. The following is the new data structure added to host:

```
struct ieee80211_nac_rssi {
    u_int8_t bssid_mac[6];
    u_int8_t client_mac[6];
    u_int8_t chan_num;
    u_int8_t client_rssi_valid;
    u_int8_t client_rssi;
};
```

This information is stored in ieee80211vap.

A new WMI command, WMI\_VDEV\_SET\_SCAN\_NAC\_RSSI\_CMDID, is introduced to send the command from host to FW side with the structure wmi\_vdev\_scan\_nac\_rssi\_config\_cmd.

```
typedef struct {
    A_UINT32 vdev_id;
    wmi_mac_addr bssid_addr;
    wmi_mac_addr client_addr;
    A_UINT32 chan_num;
    A_UINT32 action;
} wmi_vdev_scan_nac_rssi_config_cmd;
enum {
    WMI_FILTER_NAC_RSSI_ACTION_ADD      = 0x1,
    WMI_FILTER_NAC_RSSI_ACTION_REMOVE   = 0x2,
    WMI_FILTER_NAC_RSSI_ACTION_LIST    = 0x3,
}; /* NAC RSSI command */
```

If the channel is different from the home channel of athX, off-channel scan is performed.

WMI\_REQUEST\_STATS\_CMDID is leveraged to get the NAC RSSI statistics from FW. NAC RSSI statistics information is integrated in the event WMI\_UPDATE\_STATS\_EVENTID. New statistic type WMI\_REQUEST\_NAC\_RSSI\_STAT is added to represent the NAC RSSI from FW. On the host side, the NAC RSSI information is stored in vap->iv\_nac\_rssi for the VAP. If no traffic is present from the client to its associated AP (such as the client disassociated, client left, and power save values), the average RSSI is not updated.

While receiving WMI command WMI\_VDEV\_SET\_SCAN\_NAC\_RSSI\_CMDID from host, FW configures the BSSID to filter register ((RXPCU\_BSSID3\_L32, RXPCU\_BSSID3\_U16 and

RXPCU\_RX\_FILTER2, PROXY\_STA\_AD1\_SEARCH\_BSSID) to filter the frames and then the unicast frames targeted to the BSSID will be received in HW level. Client's MAC address is recorded for FW to retrieve the RSSI in SW level. Only the frames whose source MAC address matching the stored MAC address are used to retrieve the RSSI from their descriptors. The Avg RSSI is calculated with same method of associated clients and it's recorded in FW.

No mode switch occurs. If two APs work in different modes, the AP might not obtain the correct RSSI of NA.

**Table 14-4 Configure RSSI scan of NACs parameters**

| Parameter          | Format  | Description  |
|--------------------|---|--|
| rssi_nac add/del   | wlanconfig athX rssi_nac<br>add/del bssid<br>xx:xx:xx:xx:xx:xx client<br>xx:xx:xx:xx:xx:xx chan value | <p>Perform the scanning of RSSI of NACs. If the channel that is specified as the input parameter is the home channel, the BSSID and MAC address of the client are set in to firmware (FW) to receive the frames from the client to the configured BSSID. The RSSI is recorded in FW. If the channel is not equal to the home channel, off-channel scan is implemented after setting BSSID and MAC address of NAC to FW. If the channel that the client is working is not the home channel, an off-channel scan is performed.</p> <p>No special VAP is dedicated to perform the scanning of NAC RSSI.</p> <ul style="list-style-type: none"> <li>■ If same BSSID and client MAC are specified when the <b>add</b> parameter is set with the wlanconfig command, and if the channel is different, only the channel value is changed if the channel is different from home channel.</li> <li>■ The wlanconfig athX rssi_nac add command can be entered multiple times with same BSSID and client MAC continuously. When this command is entered, the first time, BSSID and client MAC are set in the FW. For all the subsequent entries of this command, only a change in channel is performed, if needed. However, if the BSSID or client MAC are different from the previous settings, an error is returned.</li> <li>■ If the BSSID and client MAC are not specified when <b>del</b> parameter is set with the wlanconfig command, an error message is displayed.</li> </ul> |
| rssi_nac show_rssi | wlanconfig athX rssi_nac<br>show_rssi   | Display the scanned RSSIs of NACs.   |

## 14.17 Channel-ranking information in ACS reports

The automatic channel selection (ACS) reporting functionality is enhanced to display the channel-ranking information. The channel-ranking capability is supported for both 2.4 GHz and 5 GHz frequency bands. Also, this feature is supported for all operating channel widths. A user can enable or disable the channel-ranking capability using iwpriv commands.

If ranking is enabled, the channel rank or grade is determined while generating ACS report. The following events occur when the request to generate ACS report is received:

1. Obtain the cached scan results

2. Generate the list of channels
3. Iterate ACS algorithm for the total number of channels supported
4. Run ACS on channel list
  - a. Retrieve best channel
  - b. Save the best channel in the list (in an ordered way)
  - c. Reduce the channel list by marking best channel in blacklist
5. Loop until the channel list is empty
6. Display the result on the user space

The wifitool is modified to display ranking information.

This section describes the capability to generate a channel score for automatic channel selection (ACS) in 2.4 GHz bands. A scored ACS report enables customers to select the best channel. A score is provided to each channel in the ACS report. A lower score indicates better channel condition, and also signifies that the channel selected by ACS algorithm must have the minimum score.

#### 14.17.1 Method of calculation of channel score

To align with ACS algorithm, the related metrics are similar to ACS parameters such as Total RSSI, Noise floor, Loading SUM, Regulatory Power and BSS number. The weighting for the metrics can be adjusted on run time for the specific scenario.

Normally, if the ACS can select the best channel, then calculate the basic score with the impact factors of the best channel. Later, compare impact factors of other channels with the best channel. If the impact factor is better than the best channel, no score will be added. If the impact factor is worse, then the difference value is multiplied by the weighting and added to the score.

If ACS cannot determine a good channel, only the BSS number is taken into account to calculate the channel score. The channel that has a smaller BSS number receives a lower score.

An example (CH10 is the best channel) is as follows:

Channel 1: NF: -97 total RSSI: 259 Loading SUM: 30 Reg Power: 30

Channel 10: NF: -95 total RSSI: 20 Loading SUM: 9 Reg Power: 30

The following is the formula used to calculate the basic score based on the impact factors of CH10:

$$\text{Basic score} = (105 - \text{NF}) * \text{NF\_WEIGHT} + \text{Total RSSI} * \text{RSSI\_WEIGHT} + \text{Loading SUM} * \text{LOAD\_WEIGHT} + (30 - \text{Regpower}) * \text{POWER\_WEIGHT}$$

The weighting value is all 10 by default.

CH10:  $(105 - 95) * 10 + 20 * 10 + 9 * 10 + (30 - 30) * 10 = 390$

The following is the formula used to calculate the score of CH 1:

Because the NF and Reg power is not worse than CH 10, it is considered as a “good value” and will not be calculated.

Score = basic score + RSSI D-Value \* RSSI\_WEIGHT + Loading SUM D-Value \* LOAD\_WEIGHT

CH 1: 390 + (259 - 20) \* 10 + (30 - 9) \* 10 = 2990

## 14.17.2 Configure and display ACS channel-ranking

To specify that ACS channel-ranking must be performed for 2 GHz bands only, enter the `iwpriv athX acs_2g_allch 1` command.

To enable ACS channel-ranking functionality, enter the `iwpriv athX acs_rank_en 1` command.

To disable ACS channel-ranking functionality, enter the `iwpriv athX acs_rank_en 0` command.

User must set correct mode and trigger ACS before generating the ACS channel-ranking report. To trigger the ACS scan, enter the `iwpriv athX chan 0` command.

The following is a sample output of an ACS report with all the channels ranked:

```
wifitool athx acsreport
```

```
Legend: SC: Secondary Channel, WR: Weather Radar, DFS: DFS Channel, HN: High Noise, RSSI: Low RSSI, CL: High Channel Load  
RP: Regulatory Power, N2G: Not selected 2G, P80X: Primary 80X80 NS80X: Only for primary 80X80, NP80X: Only for Secondary 80X80
```

The number of channels scanned for acs report is:24

| Channel   | BSS | minrss | maxrss | NF   | Ch load | spect load | sec_chan | Ranking | Unused |
|-----------|-----|--------|--------|------|---------|------------|----------|---------|--------|
| 5180( 36) | 13  | 12     | 71     | -107 | 45      | 0          | 1        | 24      | ( )    |
| 5200( 40) | 1   | 51     | 51     | -107 | 1       | 0          | 1        | 19      | ( )    |
| 5220( 44) | 1   | 37     | 37     | -107 | 3       | 0          | 1        | 16      | ( )    |
| 5240( 48) | 0   | 0      | 0      | -107 | 1       | 0          | 1        | 6       | ( )    |
| 5260( 52) | 0   | 0      | 0      | -107 | 1       | 0          | 1        | 1       | ( )    |
| 5280( 56) | 0   | 0      | 0      | -107 | 1       | 0          | 1        | 13      | ( )    |
| 5300( 60) | 0   | 0      | 0      | -106 | 1       | 0          | 1        | 17      | ( )    |
| 5320( 64) | 10  | 10     | 64     | -106 | 13      | 0          | 0        | 23      | ( )    |
| 5500(100) | 1   | 15     | 15     | -102 | 5       | 0          | 0        | 5       | ( )    |
| 5520(104) | 0   | 0      | 0      | -104 | 0       | 0          | 0        | 2       | ( )    |
| 5540(108) | 0   | 0      | 0      | -104 | 0       | 0          | 1        | 3       | ( )    |
| 5560(112) | 0   | 0      | 0      | -101 | 51      | 0          | 1        | 4       | ( )    |
| 5580(116) | 1   | 8      | 8      | -104 | 1       | 0          | 1        | 7       | ( )    |
| 5600(120) | 0   | 0      | 0      | -103 | 1       | 0          | 1        | 9       | ( )    |
| 5620(124) | 1   | 59     | 59     | -104 | 1       | 0          | 0        | 11      | ( )    |
| 5640(128) | 1   | 45     | 45     | -105 | 1       | 0          | 0        | 14      | ( )    |
| 5660(132) | 0   | 0      | 0      | -104 | 1       | 0          | 1        | 12      | ( )    |
| 5680(136) | 2   | 45     | 54     | -103 | 1       | 0          | 0        | 15      | ( )    |
| 5700(140) | 0   | 0      | 0      | -103 | 1       | 0          | 1        | 8       | ( )    |
| 5745(149) | 8   | 6      | 52     | -103 | 3       | 0          | 0        | 21      | ( )    |
| 5765(153) | 3   | 39     | 39     | -104 | 4       | 0          | 1        | 22      | ( )    |

|           |    |    |    |      |    |   |   |    |     |
|-----------|----|----|----|------|----|---|---|----|-----|
| 5180( 36) | 13 | 12 | 71 | -107 | 45 | 0 | 1 | 24 | ( ) |
| 5200( 40) | 1  | 51 | 51 | -107 | 1  | 0 | 1 | 19 | ( ) |
| 5220( 44) | 1  | 37 | 37 | -107 | 3  | 0 | 1 | 16 | ( ) |
| 5240( 48) | 0  | 0  | 0  | -107 | 1  | 0 | 1 | 6  | ( ) |
| 5260( 52) | 0  | 0  | 0  | -107 | 1  | 0 | 1 | 1  | ( ) |
| 5280( 56) | 0  | 0  | 0  | -107 | 1  | 0 | 1 | 13 | ( ) |
| 5300( 60) | 0  | 0  | 0  | -106 | 1  | 0 | 1 | 17 | ( ) |
| 5320( 64) | 10 | 10 | 64 | -106 | 13 | 0 | 0 | 23 | ( ) |
| 5500(100) | 1  | 15 | 15 | -102 | 5  | 0 | 0 | 5  | ( ) |
| 5520(104) | 0  | 0  | 0  | -104 | 0  | 0 | 0 | 2  | ( ) |
| 5540(108) | 0  | 0  | 0  | -104 | 0  | 0 | 1 | 3  | ( ) |
| 5560(112) | 0  | 0  | 0  | -101 | 51 | 0 | 1 | 4  | ( ) |
| 5580(116) | 1  | 8  | 8  | -104 | 1  | 0 | 1 | 7  | ( ) |
| 5600(120) | 0  | 0  | 0  | -103 | 1  | 0 | 1 | 9  | ( ) |
| 5620(124) | 1  | 59 | 59 | -104 | 1  | 0 | 0 | 11 | ( ) |
| 5640(128) | 1  | 45 | 45 | -105 | 1  | 0 | 0 | 14 | ( ) |
| 5660(132) | 0  | 0  | 0  | -104 | 1  | 0 | 1 | 12 | ( ) |
| 5680(136) | 2  | 45 | 54 | -103 | 1  | 0 | 0 | 15 | ( ) |
| 5700(140) | 0  | 0  | 0  | -103 | 1  | 0 | 1 | 8  | ( ) |
| 5745(149) | 8  | 6  | 52 | -103 | 3  | 0 | 0 | 21 | ( ) |
| 5765(153) | 3  | 39 | 39 | -104 | 4  | 0 | 1 | 22 | ( ) |

|           |   |    |    |      |   |   |   |    |     |
|-----------|---|----|----|------|---|---|---|----|-----|
| 5785(157) | 6 | 10 | 17 | -104 | 6 | 0 | 1 | 20 | ( ) |
| 5805(161) | 3 | 46 | 48 | -104 | 3 | 0 | 1 | 18 | ( ) |
| 5825(165) | 0 | 0  | 0  | -105 | 1 | 0 | 0 | 10 | ( ) |

The following is a sample output of an ACS report with only some of the channels ranked and the remaining channels disregarded from ranking, along with reason codes:

```
wifitool athx acsreport
Legend: SC: Secondary Channel, WR: Weather Radar, DFS: DFS Channel, HN:
High Noise, RSSI: Low RSSI, CL: High Channel Load
RP: Regulatory Power, N2G: Not selected 2G, P80X: Primary 80X80 NS80X:
Only for primary 80X80, NP80X: Only for Secondary 80X80
```

The number of channels scanned for acs report is:11

| Channel<br>Ranking | BSS<br>Unused |    | minrssi | maxrsssi | NF | Ch load | spect load | sec_chan |              |
|--------------------|---------------|----|---------|----------|----|---------|------------|----------|--------------|
| 5180( 36)          | 12            | 29 | 71      | -107     | 4  | 0       | 1          | 0        | (SC )        |
| 5220( 44)          | 1             | 37 | 37      | -107     | 1  | 0       | 1          | 0        | (SC )        |
| 5260( 52)          | 0             | 0  | -107    | 1        | 0  |         | 1          | 4        | (Random SC ) |
| 5300( 60)          | 0             | 0  | -105    | 1        | 0  |         | 1          | 0        | (SC )        |
| 5500(100)          | 1             | 15 | 15      | -102     | 4  | 0       | 0          | 1        | ( )          |
| 5540(108)          | 0             | 0  | -103    | 0        | 0  |         | 1          | 0        | (SC )        |
| 5580(116)          | 1             | 8  | 8       | -103     | 1  | 0       | 1          | 0        | (SC )        |
| 5620(124)          | 1             | 59 | 59      | -104     | 1  | 0       | 0          | 2        | ( )          |
| 5660(132)          | 0             | 0  | -104    | 1        | 0  |         | 1          | 0        | (SC )        |
| 5745(149)          | 8             | 6  | 52      | -103     | 10 | 0       | 0          | 3        | ( )          |
| 5785(157)          | 5             | 10 | 16      | -104     | 4  | 0       | 1          | 0        | (SC )        |

The ACS ranking functionality is verified for the following operating modes:

| Mode           | 2.4<br>GHz | 5<br>GHz |
|----------------|------------|----------|
| 11NAHT20       | NA         | YES      |
| 11NGHT20       | YES        | NA       |
| 11NAHT40PLUS   | NA         | YES      |
| 11NAHT40MINUS  | NA         | YES      |
| 11NGHT40PLUS   | YES        | NA       |
| 11NGHT40MINUS  | YES        | NA       |
| 11NGHT40       | YES        | NA       |
| 11NAHT40       | NA         | YES      |
| 11ACVHT20      | NA         | YES      |
| 11ACVHT40PLUS  | NA         | YES      |
| 11ACVHT40MINUS | NA         | YES      |
| 11ACVHT40      | NA         | YES      |

|              |    |     |
|--------------|----|-----|
| 11ACVHT80    | NA | YES |
| 11ACVHT160   | NA | YES |
| 11ACVHT80_80 | NA | YES |

## 14.18 Send channel statistics every second for DCS using ACFG event logs

The functionality to send periodic channel stats every 1 second and BSS channel consumption is implemented. By default, this feature is enabled.

The original DCS feature requires an iwpriv command to enable, and, in turn, sends a WMI command from host to firmware. Because this feature must be always enabled, firmware provides the service and the host calls a WMI command right after initialization to ensure this feature is enabled. The 1 second timer of DCS is used and so the DCS event is updated with the BSS software counter.

To calculate BSS channel consumption, the following hardware counters are available—Rx\_frame\_count, tx\_frame\_count and rx\_clear\_count---in cycle counts. The tx\_frame\_count includes tx packets in cycles, rx\_clear\_count includes all received packets and non-wifi interference and the rx\_frame\_count includes all received packets including other BSS, multicast and broadcast frames. To calculate our BSS consumption, we need a software counter in terms of cycles to calculate the rx packets for our BSS only. These counters are sent to host from firmware periodically every 1 second using DCS event.

The channel statistics sent for every channel on a scan is updated with the rx\_bss\_cycle\_counter and 11b data duration counter. This software counter is updated in the rx path during scan when 11b data is seen, this is needed in addition to the non-Wi-Fi interference for DCS feature. None of the software counters are reset and they only wrap around.

The host and FW changes are implemented for IPQ4019, QCA9886, QCA9984, and QCA9880 chipsets.

The my\_bss\_rx\_cycle count value is sent in the event to host with and without clients connected, the my\_bss\_rx\_cycle\_count values are sent after channel change, and the rx\_b\_mode\_data value is sent in the event to host during scan. The following is an example of a portion of the output of the event log:

```
root@OpenWrt:/# tail -f /etc/acfg_event_log
wifi0: Chan stats: frequency: 5620, noise_floor: -100, obss_utilization:
1, self_bss_utilization: 1
wifi0: Chan stats: frequency: 5620, noise_floor: -100, obss_utilization:
1, self_bss_utilization: 1
```

Self-BSS and other BSS channel utilization numbers are in percentages and these values show a difference when adequate traffic is transmitted.

## 14.19 Channel blocking

The capability to block one or more WLAN channels is implemented. This feature will be used for co-existence with LTE. Channel to be blocked depends on the LTE's operational frequency.

- There will be a ioctl to block one or more wlan channels.
- There will be a ioctl to clear the list of blocked channels.
- The blocked channel will not be selected as a primary channel when ACS chooses a channel.
- There will be a configuration option to prevent blocked channels from being used as secondary channel as well.
- There will be a configuration option to prevent blocked channels from being selected manually (using iwconfig command).

### 14.19.1 Guidelines

- Blocking channels will affect channel selections after the block list has been specified. So, if using ACS, the list of channels to be blocked has to be specified and then ACS has to be triggered. And, if manual selection is being used, then the list of channels to be blocked has to be specified first, followed by the iwconfig command to select a channel.
- ACS can select a blocked channel when all possible channels have been blocked using this command (or skipped by ACS due to other reasons).
- The check for secondary channel works only till phymode of 80MHz. It needs to be extended for 160 MHz (and 80+80 MHz). This is because the feature was implemented when 160 MHz support was not present in driver.

### 14.19.2 Specify the block list

When invoked in the manner described here, it blocks certain channels (channels 1, 2, and 3 in this example). These channels will get appended to the block list, if the block list already contained some channels. Specifying this on any VAP of a radio is sufficient.

```
wifitool athX block_acs_channel 1,2,3
```

When the following command is entered, it clears the list of blocked channels.

```
wifitool athX block_acs_channel 0
```

There is a corresponding uci setting for this. Using this allows the setting to be applied during wifi initialization sequence.

```
uci set wireless.@wifi-iface[X].channel_block_list=1,2,3
```

```
uci set wireless.@wifi-iface[X].channel_block_list=0
```

### 14.19.3 Specify the blocking mode

To prevent a channel from being allowed for manual selection (using iwconfig command).

`iwpriv wifiX acs_bmode 1`

To prevent a channel from being allowed as a secondary channel.

`iwpriv wifiX acs_bmode 2`

To prevent a channel from being allowed for manual selection (using iwconfig command) as well as prevent it from being allowed as a secondary channel.

`iwpriv wifiX acs_bmode 3`

There is a corresponding uci setting for this. Using this allows the setting to be applied during wifi initialization sequence.

`uci set wireless.wifiX.channel_block_mode=1`

`uci set wireless.wifiX.channel_block_mode=2`

`uci set wireless.wifiX.channel_block_mode=3`

### Auto Channel Selection

ACS makes a list of channels to evaluate by calling `ieee80211_acs_get_phymode_channels()`. If any channel is blocked, then we don't add channels which use this as primary to the list. We also check the blocking mode and if blocked channels cannot be used as secondary, then those channels which use these as secondary are skipped as well. `acs_is_channel_blocked()` is used to check if any channel is blocked. This makes sure that ACS will not select any blocked channel.

### Manual Channel Selection

When manually setting channel using iwconfig, we call `wlan_acs_channel_allowed()` if channel blocking mode has been set to prevent blocked channels from manual selection. This checks the primary channel (and secondary channel too if required) and returns NULL if the selected channel cannot be used. Again, `acs_is_channel_blocked()` is used to find out if a channel has been blocked. If `wlan_acs_channel_allowed()` returns NULL, the set channel ioctl would fail with error and a channel change will not happen.

## 14.19.4 Block list

### Calling from an application

```
static void block_acs_channel(const char *ifname, int argc, char *argv[])
{
#define MAX_CHANNEL 255
    int count = 0, tempb = 0, cnt = 0, i = 0, s = 0;
    u_int8_t channel[MAX_CHANNEL], valid[MAX_CHANNEL];
    char *p = NULL;
    struct iwreq iwr = { 0 };
    struct ieee80211req_athdbg req = { 0 };
    ieee80211_user_chanlist_t chanlist;
    u_int8_t *chan = NULL; /*user channel list */
    if ((argc < 4)) {
        return;
    }
}
```

```

memset(&req, 0, sizeof(struct ieee80211req_athdbg));
s = socket(AF_INET, SOCK_DGRAM, 0);
if (s < 0) {
    fprintf(stderr, "Socket open for block channel list failed\n");
    return;
}
memset(&iwr, 0, sizeof(iwr));
strncpy(iwr.ifr_name, ifname, sizeof(iwr.ifr_name));
iwr.u.data.pointer = (void *)&req;
iwr.u.data.length = (sizeof(struct ieee80211req_athdbg));
req.cmd = IEEE80211_DBGREQ_BLOCK_ACS_CHANNEL;
memset(channel, 0, MAX_CHANNEL);
p = argv[3];
while (*p != '\0') {
    while (*p != ',' && (*p != '\0')) {
        sscanf(p, "%ld", &temp);
        valid[i] = temp;
        p++;
        i++;
    }
    if (i) {
        for(cnt = 0; cnt < i; cnt++) {
            channel[count] = channel[count] + valid[cnt] * power(10,
(i- cnt -1));
        }
        count++;
        i = 0;
        if (*p == '\0')
            break;
        else
            p++; /* bypass comma */
    }
    If (count >= MAX_CHANNEL) {
        count = MAX_CHANNEL;
        break;
    }
}
if (count) {
    chan = (u_int8_t *)malloc(sizeof(u_int8_t) * (count));
    if (NULL == chan) {
        fprintf(stderr, "Memory allocation for block channel list
failed\n");
        close(s);
        return;
    }
    memcpy(chan, channel, count);
    chanlist.chan = chan;
    chanlist.n_chan = count;
    req.data.param[0] = (int)&chanlist; /* typecasting to avoid
warning */
    if (ioctl(s, IEEE80211_IOCTL_DBGREQ, &iwr) < 0) {
        fprintf(stderr, "Set block channel list failed\n");
        close(s);
        free(chan);
        return;
    }
}

```

```

    If (!chan[0])
        printf("Block channel list flushed\n");
    else {
        printf("Following channels are blocked from channel selection
algorithm\n");
        for (i = 0; i < count; i++) {
            printf("[%d]", channel[i]);
        }
        printf("\n");
    }
} else {
    fprintf(stderr, "Invalid channel list\n");
}
free(chan);
close(s);
}

```

### Blocking mode

```

static void blockmode_acs_channel(const char *ifname, int argc, char
*argv[])
{
    int s = 0;
    struct iwreq iwr = { 0 };
    unsigned int arg = 0;
    if ((argc < 4)) {
        return;
    }
    s = socket(AF_INET, SOCK_DGRAM, 0);
    if (s < 0) {
        fprintf(stderr, "Socket open for block channel list mode
failed\n");
        return;
    }
    memset(&iwr, 0, sizeof(iwr));
    memset(&arg, 0, sizeof(arg));
    strncpy(iwr.ifr_name, ifname, sizeof(iwr.ifr_name));
    iwr.u.mode = OL_ATH_PARAM_ACS_BLOCK_MODE | OL_ATH_PARAM_SHIFT; // 
ATH_PARAM_ACS_BLOCK_MODE | ATH_PARAM_SHIFT
    arg = strtoul(argv[3], NULL, 10);
    memcpy(iwr.u.name + sizeof(unsigned int), &arg, sizeof(arg));
    if (ioctl(s, ATH_HAL_IOCTL_SETPARAM, &iwr) < 0) {
        fprintf(stderr, "Set block channel list mode failed\n");
        close(s);
        return;
    }
    printf("Blocking mode is set to %u\n", arg);
    close(s);
}

```

# 15 Spectral Scan and Analysis

---

This chapter describes the essential components of the spectral scan and analysis feature, including an overview of system functionality, configuration parameters, reporting formats, host driver software architecture, host-firmware interaction (for 11ac chipsets), and supporting applications and tools.

WLAN devices share unlicensed spectrum with various commercial devices such as microwave ovens, cordless phones, video bridges, Bluetooth, and so on. The RF signals emitted by these devices can interfere with the operation of WLAN devices. Detecting the presence of these devices and classifying them is necessary to mitigate their impact. The spectral scan and analysis feature can be used for this detection and classification.

The design philosophy behind this feature is to get a rich amount of raw information from hardware, and then carry out a detailed analysis in software. This provides flexibility and extensibility. The feature is divided into two parts:

- Spectral scanning
- Spectral classification

Spectral scanning is the gathering of Fast Fourier Transform (FFT) data and some other RF measurements such as RSSI, described in further sections. It is carried out by a hardware engine as per configuration set by driver/firmware. User space tools/applications can control many of these configurations by making requests to the driver via IOCTLs. The spectral data thus gathered is sent up the receive path to the driver, and from there, to application space.

The spectral scan data is then used for spectral classification. A reference software library is provided for this. The interfering signals are classified by heuristic pattern matching using time-frequency characteristics. The library is primarily intended for use in application space and tested accordingly. This is for flexibility and avoiding compute cycles at driver priority levels. Details of the library are not discussed in this section.

It should be noted that the spectral scan and classification feature cannot help demodulate and decode information sent as part of the interfering signals (for example, video content sent by video bridges). It can only analyze the raw spectral data for detection and classification.

A demo command line application called athssd is provided. It uses the classification library. athssd feeds the spectral data gathered from the driver into the library and displays detected interferences and relevant statistics. It can also interface with external entities such as a GUI.

Spectral scan is supported in the following hardware:

- AR1022
- AR132X

- AR922X
- AR928X
- AR934X
- AR935X
- AR938X
- AR939X
- QCA955X
- AR956X\*
- AR958X
- AR959X
- QCA988X
- QCA989X
- QCA900B
- QCA6174\*\*
- QCA999x/998x

\* Spectral scan software support is not currently provided for these products.

\*\* Spectral scan support is planned for these products.

This feature has evolved since its initial inception. It has undergone major changes starting from 802.11ac chipsets. Qualcomm Technologies uses the following classification in this document in order to provide specific descriptions.

- Generation I Spectral Scan
  - Spectral scan feature as implemented on 802.11n Qualcomm Technologies chipsets; that is, prior to QCA988x/QCA989x
- Generation II Spectral Scan
  - Spectral scan feature as implemented on 802.11ac Qualcomm Technologies chipsets onwards; that is, including and succeeding AR988x/AR989x

## 15.1 Generation I spectral scan

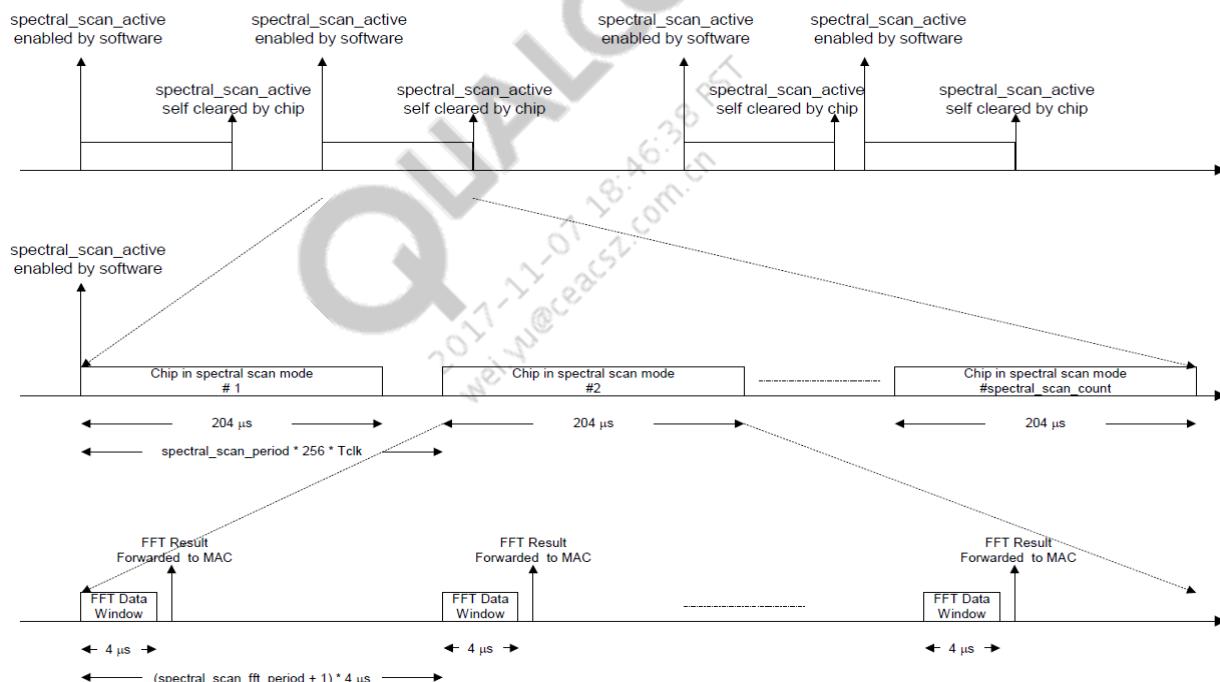
The hardware captures Spectral snapshots (FFT) of the received signal when requested by software. The rate at which the snapshots are passed to software is programmable as explained below. Setting *spectral\_scan\_active* activates the spectral scan, if *spectral\_scan\_enable* is also set (Refer to section 15.1.1 for a detailed description of these parameters).

When spectral scan is activated, the chip enters<sup>1</sup> spectral scan mode and repeatedly performs 128 point FFT over a 20 MHz bandwidth (static 20 configuration) or 256 point FFT over a 40 MHz bandwidth (dynamic 20/40 configuration) once every 4  $\mu$ s for 204  $\mu$ s. Since passing every FFT result can overwhelm the software, there is a provision to pass downsampled spectral snapshots to the software; that is, the chip passes one out of a programmable number of snapshots as specified

in *spectral\_scan\_fft\_period*. At the end of 204 µs the chip exits from spectral scan mode. However if software has enabled short reports; that is, *spectral\_scan\_short\_rpt* is 1, the chip stays in spectral scan mode for just 4 µs. This duration is sufficient to capture a single spectral snapshot. The computed FFT snapshot is passed to software and the chip exits from spectral scan mode. The chip re-enters<sup>1</sup> spectral scan mode after a programmable amount of time (specified in *spectral\_scan\_period*), performs spectral scan capture and passes the results to software as described earlier. This process is repeated *spectral\_scan\_count* times. Finally, the *spectral\_scan\_active* bit is automatically reset. Software can reactivate *spectral\_scan\_active* to retrigger the above process. However, if *spectral\_scan\_count* is programmed to 0 (128 in the case of AR922X/AR928X), the *spectral\_scan\_active* bit never self clears. That is, the spectral scan operation goes on until the software explicitly clears this bit.

Testing of the Qualcomm driver has been carried out using short reports., and thus enabling them is recommended.

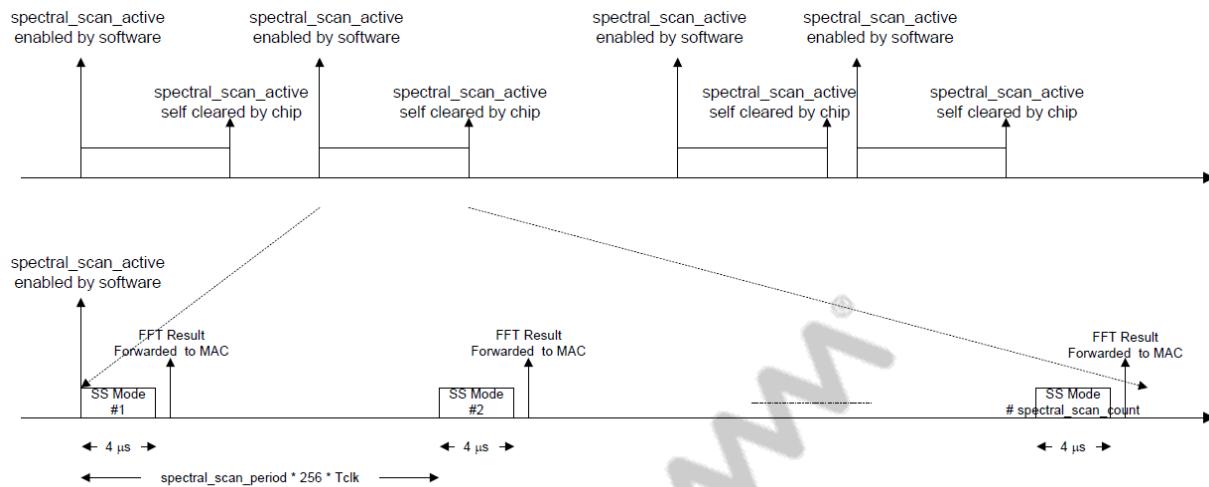
**Figure 15-1** illustrates spectral scan with *spectral\_scan\_short\_rpt* disabled (it assumes *spectral\_scan\_enable* is set to 1 and *spectral\_scan\_count* is not equal to 0/128)



**Figure 15-1 Illustration of Spectral Scan with *spectral\_scan\_short\_rpt* Disabled**

**Figure 15-2** illustrates spectral scan with *spectral\_scan\_short\_rpt* enabled (again it assumes *spectral\_scan\_enable* is set to 1 and *spectral\_scan\_count* is not equal to 0/128).

1. If the chip is currently transmitting or receiving a packet, entry into spectral scan mode is delayed until the transmission or reception of the packet is completed.
1. If the chip is currently transmitting or receiving a packet, re-entry into spectral scan mode is delayed until the transmission or reception of the packet is completed.



**Figure 15-2 Illustration of Spectral Scan with *spectral\_scan\_short\_rpt* Enabled**

### 15.1.1 Spectral scan parameters

**NOTE** The defaults given in [Table 15-1](#) refer to the values initially set by the driver.

**Table 15-1 Spectral scan parameters**

| Name                     | Width (bits)            | Default | Description   |
|--------------------------|-------------------------|---------|---|
| spectral_scan_ena        | 1                       | 0       | Enables channel frequency scanning feature. This will trigger radar style FFT data outputs, reported as radar phy errors. Reports use the 8-bits per bin spectral format.   |
| spectral_scan_active     | 1                       | 0       | Activates spectral scans. Set this bit to immediately start reporting FFT data until a timeout is reached (radar_length_max). This bit hardware self-clears after spectral_scan_count reports.  |
| spectral_scan_count      | 12 (8 in AR922X/AR928X) | 0       | Number of times the chip enters spectral scan mode before deactivating spectral scans. When set to 0 (128 in the case of AR922X/AR928X), spectral_scan_active never self clears, causing the chip to enter spectral scan mode continuously (until the software explicitly clears spectral_scan_active). |
| spectral_scan_fft_period | 4                       | 1       | Skip interval for FFT reports. Reports only the first of every (n+1) 4 μs reports. Set to '0' to forward all reports (no skip).   |
| spectral_scan_period     | 8                       | 35      | The period in which software-triggered frequency scans will automatically retrigger sending outputs to the MAC. This is used in conjunction with spectral_scan_active.<br>The units of this period are 256 * Tclk.<br>Tclk = 1/44MHz (Gmode)<br>= 1/40MHz (Amode)                                       |

**Table 15-1 Spectral scan parameters (cont.)**

| Name                    | Width (bits) | Default | Description   |
|-------------------------|--------------|---------|---|
| spectral_scan_priority  | 1            | 1       | Priority level for spectral scan triggers:<br>0: LOW PRIORITY, will trigger spectral scans if Baseband is not receiving a frame and its ADCs are in range. In other words, spectral scan has lower priority than 802.11 receive.<br>1: HIGH PRIORITY, will trigger spectral scans if ADCs are in range, and will abort (restart) from current Rx frame if spectral scan timer is ready. In other words, spectral scan has higher priority than 802.11 receive.<br>This setting is not available for AR922X/AR928X |
| spectral_scan_short_rpt | 1            | 1       | Set to report only 1 set of FFT results for timer (spectral_scan_active) triggered scans.   |

### 15.1.2 FFT report format

This section describes the FFT report format for Generation I chipsets. The spectral data is reported via the receive buffer. The descriptor is flagged as PHY\_ERROR and the error type is DFS, with the pulse\_bw\_info field set at 0x10. The length of the data reported in the receive buffer depends on the current channel width of the AP. For Static 20 mode, the hardware reports 56 FFT bins, and for Dynamic 20/40 mode, the hardware reports 128 FFT bins. The FFT bins are in linear scale, normalized, and the exponent is also available. The tables in Sections and show the format of the FFT data in both cases. The length can be deduced from the data length reported in the receive descriptor. The first 7 bytes of the receive buffer contain RSSI related information. These are as follows (one byte each).

- Primary Channel RSSI for chain 0
- Primary Channel RSSI for chain 1
- Primary Channel RSSI for chain 2
- Extension Channel RSSI for chain 0
- Extension Channel RSSI for chain 1
- Extension Channel RSSI for chain 2
- Combined Chain + Sub-Channel RSSI

The next 6 bytes of the receive buffer contain data that is not relevant to spectral scan. It may be noted that byte 0 in the tables given in sections and have an offset of 13 from the start of receive buffer.

#### Static 20 mode

**Table 15-2 Static 20 Mode**

| Byte | Description   |
|------|---|
| 0    | bin -28 magnitude[7:0] = ( $ i  +  q $ ) >> max_exp |
| 1    | bin -27 magnitude[7:0] = ( $ i  +  q $ ) >> max_exp |
| 2–53 | ...   |

**Table 15-2 Static 20 Mode**

| Byte | Description   |
|------|---|
| 54   | bin 26 magnitude[7:0] = ( $ i  +  q $ ) >> max_exp              |
| 55   | bin 27 magnitude[7:0] = ( $ i  +  q $ ) >> max_exp              |
| 56   | [7:0]: all bins {max_magnitude[1:0], bitmap_weight[5:0]}        |
| 57   | [7:0]: all bins max_magnitude[9:2]                              |
| 58   | [7:0]: all bins {max_index[5:0], max_magnitude[11:10]}          |
| 59   | [3:0]: max_exp (shift amount to size max bin to 8-bit unsigned) |

**Dynamic 20/40 mode****Table 15-3 Dynamic 20/40 Mode**

| Byte  | Description   |
|-------|---|
| 0     | bin -64 magnitude[7:0] = ( $ i  +  q $ ) >> max_exp             |
| 1     | bin -63 magnitude[7:0] = ( $ i  +  q $ ) >> max_exp             |
| 2–125 | ...   |
| 126   | bin 62 magnitude[7:0] = ( $ i  +  q $ ) >> max_exp              |
| 127   | bin 63 magnitude[7:0] = ( $ i  +  q $ ) >> max_exp              |
| 128   | [7:0]: lower bins {max_magnitude[1:0], bitmap_weight[5:0]}      |
| 129   | [7:0]: lower bins max_magnitude[9:2]                            |
| 130   | [7:0]: lower bins {max_index[5:0], max_magnitude[11:10]}        |
| 131   | [7:0]: upper bins {max_magnitude[1:0], bitmap_weight[5:0]}      |
| 132   | [7:0]: upper bins max_magnitude[9:2]                            |
| 133   | [7:0]: upper bins {max_index[5:0], max_magnitude[11:10]}        |
| 134   | [3:0]: max_exp (shift amount to size max bin to 8-bit unsigned) |

## 15.2 Generation II spectral scans

### 15.2.1 Generation II enhancements

The following enhancements are made for Generation II spectral scan. Refer to for details of individual settings.

#### 15.2.1.1 Dedicated FFT engine

AR988x/AR989x has a dedicated FFT Engine that is not shared with WLAN traffic. It is arbitrated with radar detection.

### 15.2.1.2 Programmable spectral scan FFT size

The FFT size used for spectral scanning is programmable between 4-point FFT and 512-point FFT for each bandwidth mode using the configuration parameter *spectral\_scan\_fft\_size*. This yields an increased FFT resolution. This is illustrated in the below table (which is for the default setting of an internal baseband clock mode, *ovsamp\_clk\_mode*=0).

**Table 15-4 FFT Size vs. Bandwidth Modes**

| Operational Bandwidth | FFT Size |        |         |          |          |           |            |            |
|-----------------------|----------|--------|---------|----------|----------|-----------|------------|------------|
|                       | 4        | 8      | 16      | 32       | 64       | 128       | 256        | 512        |
| 20 MHz                | 10 MHz   | 5 MHz  | 2.5 MHz | 1.25 MHz | 625 kHz  | 312.5 kHz | 156.25 kHz | 78.125 kHz |
| 40 MHz                | 20 MHz   | 10 MHz | 5 MHz   | 2.5 MHz  | 1.25 MHz | 625 kHz   | 312.5 kHz  | 156.25 kHz |
| 80 MHz                | 40 MHz   | 20 MHz | 10 MHz  | 5 MHz    | 2.5 MHz  | 1.25 MHz  | 625 kHz    | 312.5 kHz  |

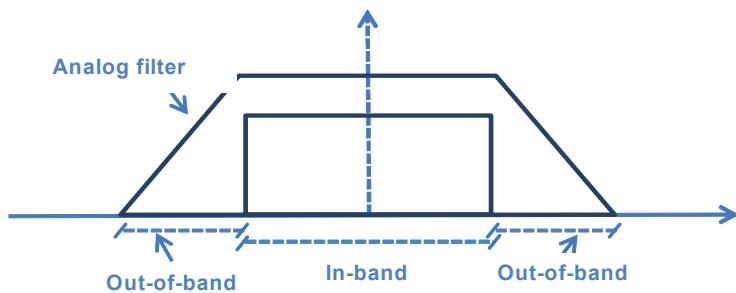
### 15.2.1.3 Ability to Perform Spectral Scan over the Entire Operational Bandwidth

Spectral scan can be performed over all modes of operational bandwidth; that is, 20 MHz, 40 MHz and 80 MHz. Even though an operational bandwidth of 80 MHz is not allowed in the 2.4 GHz band, the entire 2.4 GHz band can be scanned in one capture in sniffer mode.

**NOTE** Software support for 80 MHz coverage in 2.4 GHz has not yet been added as of this release.

### 15.2.1.4 Spectral Scan FFT computed at Full ADC Rate

Spectral scan FFTs are computed at full ADC sampling rate using a dedicated FFT engine. This allows reporting in band as well as out of band bins as shown in the figure below. Note that reported out of band bin magnitude will not be accurate due to attenuation from analog filtering. This can be corrected in software if needed.



**Figure 15-3 Spectral Scan FFT computed at Full ADC Rate**

### 15.2.1.5 Reporting Format Options for FFT Bin Magnitudes

Spectral scan FFT bin magnitudes can be reported in linear, log or absolute (antenna input referred) dBm formats based on programmable register settings (*spectral\_scan\_pwr\_format* and *spectral\_scan\_dBm\_adj*).

### 15.2.1.6 Improved Reporting Precision

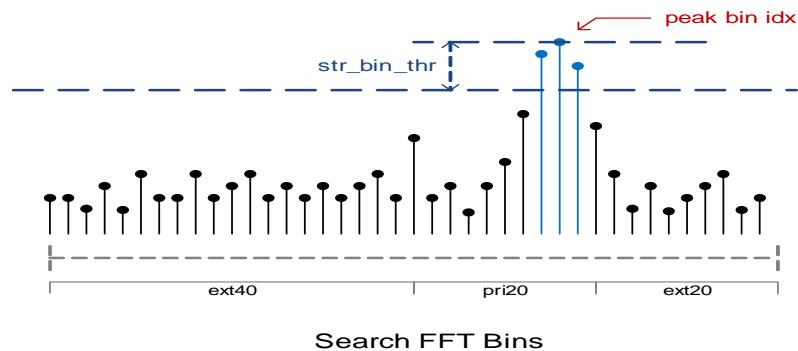
The Spectral scan FFT bin magnitudes can be reported with an improved precision of  $\frac{1}{2}$  dB steps with *spectral\_scan\_bin\_scale* set to 0.

### 15.2.1.7 Spectral scan Report Filtering in Hardware

Spectral scan reports can optionally get filtered from reporting to software if either the RSSI is too low or the signal is deemed wideband.

- **RSSI Filtering:** If the ADC power is below the configuration parameter *spectral\_scan\_rssi\_thr*, the spectral scan request will get serviced without an FFT computation. These reports will only contain a spectral scan summary report noting that the spectral scan request was serviced. These reports can be tagged with an alternate (ignored) Phy Error code EXT\_BLOCKER (0x24 for AR9888, and 0x01000000 for AR900B), by setting *spectral\_scan\_rssi\_rpt\_mode*.
- **Wideband Filtering:** If the FFT has a strong bin count greater than the configuration parameter *spectral\_scan\_nb\_tone\_thr*, the spectral scan is deemed wideband. These reports can be tagged with an alternate (ignored) Phy Error code EXT\_BLOCKER (0x24 for AR9888, and 0x01000000 for AR900B), by setting *spectral\_scan\_wb\_rpt\_mode*.

Each completed FFT includes a wideband/narrowband classification in each sub-channel. This is done by comparing the relative magnitude of each bin to the peak in-band bin magnitude.



**Figure 15-4 Determination of Strong Bins for Wideband/Narrowband Classification during Spectral FFT**

The relative bandwidth of the signals in each sub-channel is quantified by counting the number of bins in each sub-channel whose magnitudes are larger than a threshold specified by *spectral\_scan\_str\_bin\_thr*. For linear format, this threshold is equal to the peak in-band bin linear magnitude multiplied by *spectral\_scan\_str\_bin\_thr*/8, whereas for dB format, this threshold is equal to the

peak in-band bin magnitude in dB minus *spectral\_scan\_str\_bin\_thr*. These bins are defined (whose magnitudes are larger than this threshold) as **strong bins** (shown in blue). Counting the strong bins in each sub-channel determines whether the signal is narrowband (most interferer signals), wideband (typically WLAN packets/DSSS signals), or somewhere in between. Some other narrowband check requirements:

- If a sub-channel is declared narrowband based on the strong bin count, it must also contain the peak bin index. All other sub-channels which do not contain the peak bin index are declared not narrowband.
- If any sub-channel has a strong bin count that classifies as wideband (that is, extension blocker), then no sub-channel will be declared narrowband.
- This check uses the programmable thresholds *nb\_tone\_thr\_radar\_blk* (not currently exposed to end users) and *spectral\_scan\_str\_bin\_thr*.

#### 15.2.1.8 Reporting of New Fields

Reporting of new fields include current Rx gain, recent rfsat flag, average power, base power, out-of-band flag, FFT chain index, spectral scan group index, and so on. See [Search FFT Report \(Optional\)](#) and [for a detailed list of reported fields](#).

#### 15.2.1.9 Ability to Specify the Input Source Rx Chain for FFTs

Spectral scan FFTs can be computed using the input from any of the Rx chains as specified by the configuration parameter *spectral\_scan\_chn\_mask*. In previous chips, only Rx chain 0 can be used.

#### 15.2.1.10 Spectral Scan Group Index Reporting

Each Spectral scan report includes the index of the current spectral scan capture. This is useful for finding out missed or filtered out captures.

#### 15.2.1.11 Ability to Trigger FFT without Waiting for Gain to Settle

This feature can be enabled by setting the configuration parameter *spectral\_scan\_gc\_ena* to 0. This is useful for RF jammer detection (not yet included in software classification library).

#### 15.2.1.12 Improved Spectral Scan Sequencing and Request Servicing

There are three different priority levels that can be set for spectral scan request servicing as desired by the user: low, medium, and high priority. Spectral scan requests are triggered periodically in hardware. These requests get serviced according to the priority settings of spectral scan:

**Low Priority:** This is the default mode, where spectral scans only get serviced if there is no current Tx or Rx frame active, and the ADC signal sizing is stable and in range (that is, no coarse gain adjustments queued).

**Medium Priority:** If *spectral\_scan\_priority* is set, spectral scans get serviced as long as no current Tx or Rx frame is active. ADC signal sizing does not need to be stable, so this will trigger Spectral scans even on rapidly transitioning signals (for example, jammers).

**High Priority:** If both *spectral\_scan\_priority* and *spectral\_scan\_restart\_ena* are set, Spectral scans get serviced as long as no current Tx frames are active. Active Rx frames will get aborted to service spectral scans with the aborted frame sending a restart Phy Error.

### 15.2.1.13 Programmable Timer to Delay Spectral Scan Triggers

This programmable threshold is specified by the parameter *spectral\_scan\_init\_delay*. It is used to disallow spectral scan triggers after Tx/Rx packets by setting this value to roughly SIFS time period or greater. This delay timer counts in units of 0.25  $\mu$ s, and is only used after valid (Non-Phy\_Error) Rx/Tx frames.

## 15.2.2 Known Limitations

Listed below are some of the current known limitations in AR988x/AR989x Spectral scan architecture.

- No Spectral scans during Tx Frames
- No Spectral scans during Rx Frames: currently the Rx frames are aborted and a spectral scan is started.

## 15.2.3 QCA999x/998x Spectral Scan Event Reporting

### 15.2.3.1 Initiating Spectral Scan Reporting

Qualcomm Technologies provides the information below on steps required to request hardware to initiate spectral scan reporting. But note that these steps are already taken care of in firmware, in the Spectral WHAL module. Refer to [} \\_\\_ATTRIB\\_PACK\\_SPECTRAL\\_SAMP\\_DATA](#); for details of host-firmware interaction required to start spectral scans. The following information is provided only for reference.

Spectral scans are initiated by setting the active bit of the spectral scan control. The hardware is capable of re-triggering scan requests based on the timer interval defined in the configuration registers. Spectral scan requests are typically deferred if there is an active Tx/Rx packet, or if the ADC signal sizing does not look stable (see priority settings in ). The total number of scans performed is defined by the *spectral\_scan\_count* parameter. A value of ‘0’ indicates an infinite count.

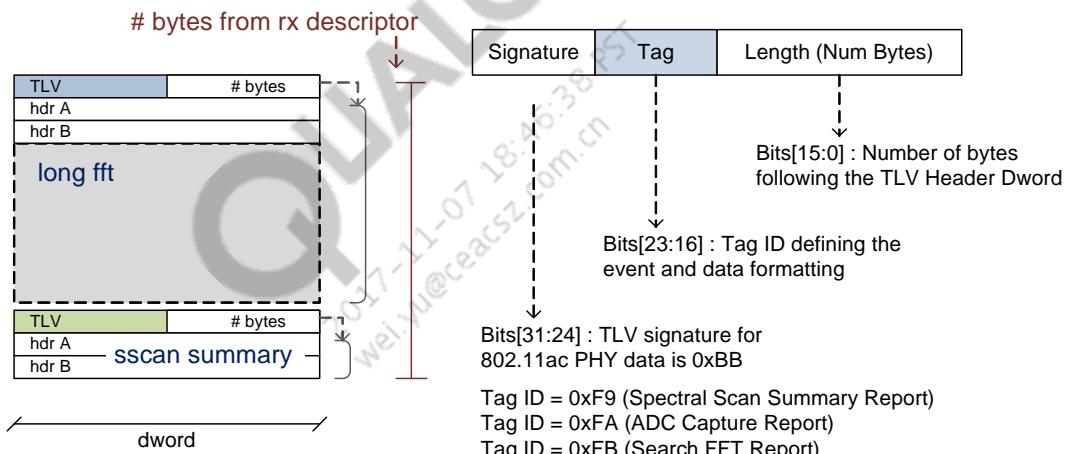
Main setup steps:

1. Ensure *spectral\_scan\_ena* and *spectral\_scan\_active* are both set to ‘0’.
2. Set *spectral\_scan\_ena* to ‘1’.
3. Write the number of FFT points required to *spectral\_scan\_fft\_size*.  
Write the number of Spectral scan reports desired to *spectral\_scan\_count*.  
Write the spectral scan request interval desired to *spectral\_scan\_period*.
4. Set the request priority through *spectral\_scan\_priority* and *spectral\_scan\_restart\_ena*.

5. Set the reporting format desired for the FFT data (see [for configuration fields](#)).  
Set the filtering thresholds and filtered report format (see [for main configuration fields](#)).
  6. When the spectral scan count value reaches *spectral\_scan\_count*, the hardware will wrap up operations and self-clear the *spectral\_scan\_active* bit.  
If disabling spectral scan requests before the hardware count is reached is desired, write ‘0’ to *spectral\_scan\_active*, then (optionally) set *spectral\_scan\_ena* to ‘0’ after an FFT processing period of up to 20  $\mu$ s.

### 15.2.3.2 Rx Packet

The Rx Packet data payload from the MAC to firmware can contain two different types of encapsulated payloads each of which is delineated by a Tag Length Vector (TLV) Dword that prefaces the data grouping. The TLVs are sent as-is by firmware to the host. The TLV Dword header defines the format of the data to follow and indicates the number of bytes associated with the data grouping. Two TLV tags are valid for Spectral scan reporting: **Search FFT Report (0xFB)** and **Spectral Scan Summary Report (0xF9)**.



**Figure 15-5 Overview of Spectral TLV Formats**

Each TLV packet is guaranteed to complete; so the number of bytes indicated in a TLV will always follow a TLV header (that is, no incomplete FFT reports).

## **Search FFT Report (Optional)**

The tag ID for Search FFT Reports is **0xFB**. This TLV has a data payload of N+8 bytes, where N represents the number of FFT bins that are getting reported and 8 represents the number of summary header bytes. Reporting formats for all Search FFT TLV reports is defined by the configuration parameters *spectral\_scan\_rpt\_mode*, *spectral\_scan\_pwr\_format*, and *spectral\_scan\_dBm\_adj*.

**Table 15-5 Search FFT Report**

| <b>Field</b>    | <b>Dword</b> | <b>Index</b> | <b>Value</b> | <b>Description</b>  |
|-----------------|--------------|--------------|--------------|---|
| TLV_HDR_SIG     | 0            | bit[31:24]   | 0xBB         | Signature for Baseband PHY generated TLV packets  |
| TLV_HDR_TAG     | 0            | bit[23:16]   | 0xFB         | Tag ID for Search FFT Report  |
| TLV_HDR_LENGTH  | 0            | bits[15:0]   | N+8          | Number of bytes following the TLV header (Dword 1 – Dword (N+8)/4)  |
| total_gain_db   | 1            | bit[31:23]   |              | Total radio gain index at time of FFT (dB step)   |
| base_pwr_db     | 1            | bit[22:14]   |              | dB offset used to convert FFT log2 bin magnitude to dBm   |
| fft_chn_idx     | 1            | bits[13:12]  |              | Rx chain index used (internal BB chain) for this FFT  |
| peak_sidx       | 1            | bits[11:0]   |              | Index of peak magnitude bin (signed). Sidx==0 is DC.<br>Bin resolution is:<br>$40*(2^{dyn\_bw})/(Npts\ FFT)\ MHz$<br>$44*(2^{dyn\_bw})/(Npts\ FFT)\ MHz\ \{ovsamp\_clk\_mode=1\}$ |
| relpwr_db       | 2            | bits[31:26]  |              | In-band / Out-band power ratio (dB step) for this FFT   |
| avgpwr_db       | 2            | bits[25:18]  |              | In-band bins summation used for relpwr_db computation.  |
| peak_mag        | 2            | bits[17:8]   |              | Magnitude of peak bin (linear or dB)  |
| num_str_bins_ib | 2            | bits[7:0]    |              | Number of “strong” in-band bins as defined by <str_bin_thr_radar> (not currently exposed to end users)  |
| bin_mags        | ...          | ...          |              | ...   |
| bin_mag(N-1)    | N/4+2        | bits[31:24]  |              | Magnitude of FFT bin (Max neg IDX first... Max pos IDX last)  |
| bin_mag(N-2)    | N/4+2        | bits[23:16]  |              | Magnitude of FFT bin (Max neg IDX first... Max pos IDX last)  |
| bin_mag(N-3)    | N/4+2        | bits[15:8]   |              | Magnitude of FFT bin (Max neg IDX first... Max pos IDX last)  |
| bin_mag(N-4)    | N/4+2        | bits[7:0]    |              | Magnitude of FFT bin (Max neg IDX first... Max pos IDX last)  |

**Table 15-6 Search FFT report (QCA9984/QCA9888 160/80+80 MHz chipset)**

| <b>Field</b>   | <b>Dword</b> | <b>Index</b> | <b>Value</b> | <b>Description</b>  |
|----------------|--------------|--------------|--------------|---|
| TLV_HDR_SIG    | 0            | bit[31:24]   | 0xBB         | Signature for Baseband PHY generated TLV packets  |
| TLV_HDR_TAG    | 0            | bit[23:16]   | 0xFB         | Tag ID for Search FFT Report  |
| TLV_HDR_LENGTH | 0            | bits[15:0]   | N+12         | Number of bytes following the TLV header (Dword 1 – Dword (N+12)/4)   |
| total_gain_db  | 1            | bit[31:23]   |              | Total radio gain index at time of FFT (dB step)   |
| base_pwr_db    | 1            | bit[22:14]   |              | dB offset used to convert FFT log2 bin magnitude to dBm   |
| fft_chn_idx    | 1            | bits[13:12]  |              | Rx chain index used (internal BB chain) for this FFT  |
| peak_sidx      | 1            | bits[11:0]   |              | Index of peak magnitude bin (signed). Sidx==0 is DC.<br>Bin resolution is:<br>$40*(2^{dyn\_bw})/(Npts\ FFT)\ MHz$<br>$44*(2^{dyn\_bw})/(Npts\ FFT)\ MHz\ \{ovsamp\_clk\_mode=1\}$ |
| relpwr_db      | 2            | bits[31:26]  |              | In-band / Out-band power ratio (dB step) for this FFT   |

**Table 15-6 Search FFT report (QCA9984/QCA9888 160/80+80 MHz chipset)**

| Field             | Dword | Index       | Value | Description  |
|-------------------|-------|-------------|-------|--|
| avgpwr_db         | 2     | bits[25:18] |       | In-band bins summation used for relpwr_db computation.               |
| peak_mag          | 2     | bits[17:8]  |       | Magnitude of peak bin (linear or dB)                                 |
| num_str_bins_ib   | 2     | bits[7:0]   |       | Number of “strong” in-band bins as defined by <str_bin_thr_radar>    |
| <b>segment_id</b> | 3     | bits[0]     |       | The segment ID used during 80+80 mode.<br>0 = primary, 1 = secondary |
| bin_mags          |       |             |       |  |
| bin_mag(N-1)      | N/4+3 | bits[31:24] |       | Magnitude of FFT bin (Max neg IDX first... Max pos IDX last)         |
| bin_mag(N-2)      | N/4+3 | bits[23:16] |       | Magnitude of FFT bin (Max neg IDX first... Max pos IDX last)         |
| bin_mag(N-3)      | N/4+3 | bits[15:8]  |       | Magnitude of FFT bin (Max neg IDX first... Max pos IDX last)         |
| bin_mag(N-4)      | N/4+3 | bits[7:0]   |       | Magnitude of FFT bin (Max neg IDX first... Max pos IDX last)         |

### Spectral Scan Summary Report

The tag ID for Spectral Scan Summary Reports is **0xF9**. This TLV has a data payload of 8 bytes (20 bytes in the case of the QCA9984/QCA9888 chipset), and contains summary information pertaining to the overall spectral scan trigger.

**Table 15-7 Spectral Scan Summary Report (QCA999x/998x)**

| Field           | Dword | Index       | Value | Description   |
|-----------------|-------|-------------|-------|---|
| TLV_HDR_SIG     | 0     | bit[31:24]  | 0xBB  | Signature for Baseband PHY generated TLV packets  |
| TLV_HDR_TAG     | 0     | bit[23:16]  | 0xF9  | Tag ID for spectral scan Summary Report   |
| TLV_HDR_LENGTH  | 0     | bits[15:0]  | 8     | Number of bytes following the TLV header (Dword 1,2)  |
| agc_mb_gain     | 1     | bits[30:24] |       | Mixer-Baseband radio gain index in ½ dB step.   |
| sscan_gidx      | 1     | bits[23:16] |       | Spectral scan group index. This index increments by one for each fully serviced scan request.   |
| agc_total_gain  | 1     | bits[9:0]   |       | Total radio gain index in ½ dB step.  |
| recent_rfstat   | 2     | bit[31]     |       | This bit is set if a recent RF peak detector flag triggered an RF saturation event. The definition of “recent” is determined by the setting of configuration parameter <rfsat_2_add_rfgain_delay> (not currently exposed to end users). |
| sscan_ob_flag   | 2     | bit[30]     |       | Flag, indicating that the signal was determined to be out-of-band.  |
| sscan_nb_mask   | 2     | bits[29:22] |       | Per sub-channel signal narrowband indication flags. Bit[29] is the most negative sub-channel (-70MHz) and bit[22] is the most positive sub-channel (+70MHz). Bit[25] (+10MHz) is the default mask if BB is in static20 mode.            |
| sscan_peak_mag  | 2     | bits[21:12] |       | Magnitude of peak bin (linear, dB, or dBm)  |
| sscan_peak_sidx | 2     | bits[11:0]  |       | Index of peak in-band bin magnitude (signed). Sidx==0 is DC. Bin resolution is:<br>$40*(2^{dyn\_bw})/(Npts\ FFT)\ MHz$<br>$44*(2^{dyn\_bw})/(Npts\ FFT)\ MHz\ \{ovsamp\_clk\_mode=1\}$  |

**Table 15-8 Spectral Scan Summary Report (QCA9984/QCA9888 160/80+80 MHz chipset)**

| Field            | Dword | Index       | Value | Description   |
|------------------|-------|-------------|-------|---|
| TLV_HDR_SIG      | 0     | bit[31:24]  | 0xBB  | Signature for Baseband PHY generated TLV packets  |
|                  | 0     |             |       |   |
| TLV_HDR_TAGq     | 0     | bit[23:16]  | 0xF9  | Tag ID for Spectral Scan Summary Report   |
| TLV_HDR_LENGTH   | 1     | bits[15:0]  | 8     | Number of bytes following the TLV header (Dword 1,2)  |
| recent_rfsat     | 1     | bit[8]      |       | This bit is set if a recent RF peak detector flag triggered an RF saturation event. The definition of "recent" is determined by the setting of configuration parameter <rfsat_2_add_rfgain_delay>.                            |
| sscan_gidx       | 2     | bits[7:0]   |       | Spectral Scan group index. This index increments by one for each fully serviced scan request.   |
| sscan_ob_flag0   | 2     | bit[17]     |       | Flag, indicating that the signal was determined to be out-of-band.  |
| agc_mb_gain0     | 2     | bits[16:10] |       | Mixer-Baseband radio gain index in ½ dB step.   |
| agc_total_gain0  | 3     | bits[9:0]   |       | Total radio gain index in ½ dB step.  |
| sscan_nb_mask0   | 3     | bits[29:22] |       | Per sub-channel signal narrow band indication flags. Bit[29] is the most negative sub-channel (-70MHz) and bit[22] is the most positive sub-channel (+70MHz). Bit[25] (+10MHz) is the default mask if BB is in static20 mode. |
| sscan_peak_mag0  | 3     | bits[21:12] |       | Magnitude of peak bin (linear, dB, or dBm)  |
| sscan_peak_sidx0 | 4     | bits[11:0]  |       | Index of bin with peak in-band magnitude (signed). Sidx==0 is DC.<br>Bin resolution is:<br>$40 * (2^{\text{dyn\_bw}}) / (\text{Npts FFT})$ MHz<br>$44 * (2^{\text{dyn\_bw}}) / (\text{Npts FFT})$ MHz {ovsamp_clk_mode=1}     |
| sscan_ob_flag1   | 4     | bit[17]     |       | Flag, indicating that the signal was determined to be out-of-band.  |
| agc_mb_gain1     | 4     | bits[16:10] |       | Mixer-Baseband radio gain index in ½ dB step.   |
| agc_total_gain1  | 5     | bits[9:0]   |       | Total radio gain index in ½ dB step.  |
| sscan_nb_mask1   | 5     | bits[29:22] |       | Per sub-channel signal narrow band indication flags. Bit[29] is the most negative sub-channel (-70MHz) and bit[22] is the most positive sub-channel (+70MHz). Bit[25] (+10MHz) is the default mask if BB is in static20 mode. |
| sscan_peak_mag1  | 5     | bits[21:12] |       | Magnitude of peak bin (linear, dB, or dBm)  |
| sscan_peak_sidx1 | 5     | bits[11:0]  |       | Index of bin with peak in-band magnitude (signed). Sidx==0 is DC.<br>Bin resolution is:<br>$40 * (2^{\text{dyn\_bw}}) / (\text{Npts FFT})$ MHz<br>$44 * (2^{\text{dyn\_bw}}) / (\text{Npts FFT})$ MHz {ovsamp_clk_mode=1}     |

## 15.2.4 Generation II Spectral Scan Configuration Registers

**Table 15-9** lists the spectral scan configuration registers exposed to end users. Note that the defaults mentioned are those initially configured by the host driver, as of this release. These defaults may not always correspond to the initial defaults used by hardware.

**Table 15-9 Generation II Spectral Scan Configuration Register**

| Name                          | Width (bits) | Value | Description  |
|-------------------------------|--------------|-------|--|
| spectral_scan_ena             | 1            | 0     | Set to 1 to enable spectral scan feature. Logic remains idle until the active bit described next is also set.  |
| specral_scan_active           | 1            | 0     | Set to 1 to start triggering spectral scan reports. Scans are queued up according to a time interval defined by <spectral_scan_period>. The queues have the lowest AGC (signal detection) priority by default. So reports will typically trigger during idle gain change activity. This bit will stay asserted while there are pending scans, and will (HW) self-clear after <spectral_scan_count> reports are complete.                                 |
| spectral_scan_fft_size        | 4            | 7     | Defines the number of FFT data points to compute according to: num_fft_pts = $2^{<\text{spectral\_scan\_ftt\_size}>}$ , for example, 512-pt FFT => $2^9$ .   |
| spectral_scan_period          | 8            | 35    | Defines the period in which spectral scans will re-trigger sending reports to the MAC. Used in conjunction with <spectral_scan_active>. Period increment resolution is $256 \times \text{Tclk}$ where Tclk is 1/40MHz (1/44MHz if ovsamp_clk_mode ==1).  |
| spectral_scan_count           | 12           | 0     | Number of spectral scan reports to send before de-assertion of <spectral_scan_active>. If set to 0, <spectral_scan_active> will not self-clear.  |
| spectral_scan_priority        | 1            | 1     | Priority level for spectral scan triggers:<br>Set to 0 for LOW PRIORITY, i.e. will trigger spectral scans if BB is not receiving / transmitting a frame and ADCs are in range (no coarse gain adjustments queued).<br>Set to 1 for MEDIUM PRIORITY, i.e. will trigger spectral scans if BB is not receiving / transmitting a frame. Will abort (restart) from current receive frame if a spectral scan is queued and <spectral_scan_restart_ena> is set. |
| spectral_scan_gc_ena          | 1            | 1     | Set to 1 to allow the AGC to perform a targeted gain change before starting the spectral scan FFT.   |
| spectral_scan_restart_ena     | 1            | 0     | Set to 1 to enable abort of receive frames when a spectral scan is queued and <spectral_scan_priority> is also set. This is the HIGH PRIORITY mode for spectral scan.  |
| spectral_scan_noise_floor_ref | 8            | -96   | Noise floor reference number for the calculation of bin power (signed dBm).  |
| specral_scan_init_delay       | 7            | 80    | Disallow spectral scan triggers after TX/RX packets by setting this delay value to roughly SIFS time period or greater. Delay timer counts in units of 0.25us, and is only used after valid (Non-Phy_Error) RX/TX frames.  |

**Table 15-9 Generation II Spectral Scan Configuration Register (cont.)**

| Name                        | Width (bits) | Value | Description  |
|-----------------------------|--------------|-------|--|
| spectral_scan_nb_tone_thr   | 8            | 12    | <p>Number of strong bins (inclusive) per sub-channel, below which a signal is declared a narrow band tone. This parameter is normalized for sub-channel bin resolutions of ~312.5kHz (64 bins per sub-channel, ovsamp_clk=0). Scale thresholds for other FFT sizes proportionately, i.e. set &lt;spectral_scan_nb_tone_thr&gt; = (no. of strong bins desired * 64/Npt_ib), where Npt_ib= (FFT size/(2^(dyn_BW+1))). For example:</p> <p>To achieve a value of 5 for a 32-pt static20 FFT (1 sub-channel of 16 bins), set spectral_scan_nb_tone_thr=20</p> <p>To achieve a value of 5 for a 128-pt dynamic80 FFT(4-sub-channels of 16 bins), set spectral_scan_nb_tone_thr=20</p> <p>To achieve a value of 5 for a 512-pt dynamic80 FFT(4-sub-channels of 64 bins), set spectral_scan_nb_tone_thr=5</p> |
| spectral_scan_str_bin_thr   | 6            | 8     | <p>Used to specify the threshold over which a bin is declared strong (for spectral scan bandwidth analysis).</p> <p>When fft_pwr_format==0, this scalar is divided by 8 for a linear comparison threshold, i.e; threshold above which a bin is declared strong =max_bin_mag*spectral_scan_str_bin_thr/8.</p> <p>When fft_pwr_format==1, this is a dB-step backoff threshold, i.e.; threshold above which a bin is declared strong=max_bin_mag (in dB) -spectral_scan_str_bin_thr (in dB).</p> <p>Decrease to reject more bins that are not very strong.</p>  |
| spectral_scan_wb_rpt_mode   | 1            | 0     | Set to 1 to report spectral scans as EXT_BLOCKER (Phy_Error=0x24 for AR9888, and 0x01000000 for AR900B) if none of the sub-channels are deemed narrow band. If set to 0, the narrow band classification will be ignored and the spectral scan will get reported with the default error code (Phy_Error=38).  |
| spectral_scan_rssi_rpt_mode | 1            | 0     | Set to report spectral scans as EXT_BLOCKER (Phy_Error=0x24 for AR9888, and 0x01000000 for AR900B) if the ADC RSSI is below the threshold <spectral_scan_rssi_thr>. If set to 0, these spectral scans (with low RSSI) will get reported with the default error code (Phy_Error=38). In either case, if the ADC RSSI is below <spectral_scan_rssi_thr>, then FFTs will not trigger, but timestamps and summaries gets reported.   |
| spectral_scan_rssi_thr      | 8            | 0xf0  | ADC RSSI must be greater than or equal to this threshold (signed dB) to ensure spectral scan reporting with normal error code (Phy_Error=38) (see <spectral_scan_rssi_rpt_mode> above). When RSSI is below this threshold, the FFT is not computed but the event is still timestamped and reported. The Phy Error code used in this case is defined by the configuration bit <spectral_scan_rssi_rpt_mode>.  |
| spectral_scan_pwr_format    | 1            | 0     | Format of frequency bin magnitude for spectral scan triggered FFTs:<br>0: linear magnitude<br>1: log magnitude ( $20 \cdot \log_{10}(\text{lin\_mag})$ ). Magnitudes are reported in $\frac{1}{2}$ dB steps.   |

**Table 15-9 Generation II Spectral Scan Configuration Register (cont.)**

| Name                    | Width (bits) | Value | Description   |
|-------------------------|--------------|-------|---|
| spectral_scan_rpt_mode  | 2            | 2     | Format of FFT report to software for spectral scan triggered FFTs.<br>0: No FFT report (only spectral scan summary report)<br>1: 2-dword summary of metrics for each completed FFT + spectral scan summary report<br>2: 2-dword summary of metrics for each completed FFT + 1x-oversampled bins (in-band) per FFT + spectral scan summary report In the case of QCA9984/QCA9888, 8 additional bins are reported, 4 bins to the left and right of the band-edge<br>3: 2-dword summary of metrics for each completed FFT + 2x-oversampled bins (all) per FFT + spectral scan summary report |
| spectral_scan_bin_scale | 2            | 1     | Number of LSBs to shift out in order to scale the FFT bins. Used for spectral scan triggered FFTs. Set to at least 1 when using dBm format reporting (precision is dB steps in this case). Precision is $\frac{1}{2}$ dB steps when set to 0.   |
| spectral_scan_dBm_adj   | 1            | 1     | Set to 1 (with spectral_scan_pwr_format=1), to report bin magnitudes in dBm power. This is computed using the noisefloor calibration results. Due to precision constraints, <spectral_scan_bin_scale> should also be set to 1. Precision is in dB steps in this case.   |
| spectral_scan_chn_mask  | 4            | 0x1   | Per chain enable mask to select input ADC for search FFT. If more than one chain is enabled, the max valid chain is used. LSB corresponds to Chain 0. In previous chips, only logic Chain 0 was used.   |

## 15.3 Software Architecture

This section describes the host driver architecture, and for Generation II, firmware functionality overview and host-firmware interaction.

### 15.3.1 Spectral Host Driver

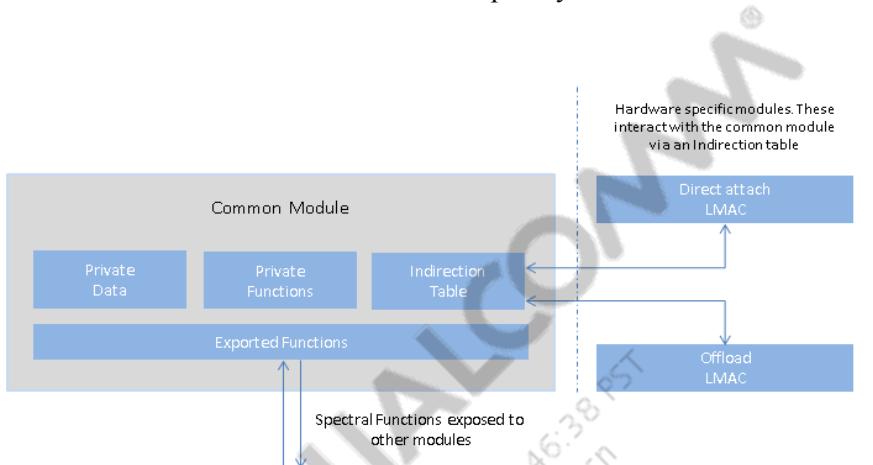
#### Files

```
drivers/wlan_modules/lmac/spectral/spectral.c
drivers/wlan_modules/lmac/spectral/spectral.h
drivers/wlan_modules/lmac/spectral/spectral_cmds.c
drivers/wlan_modules/lmac/spectral/spectral_ioctl.h
drivers/wlan_modules/lmac/spectral/spectral_netlink.c
drivers/wlan_modules/lmac/spectral/spectral_phyerr.c
drivers/wlan_modules/lmac/spectral/spectral_process_data.c
drivers/wlan_modules/lmac/spectral/spectral_samp.c
drivers/wlan_modules/lmac/spectral/spectral_types.h
drivers/wlan_modules/lmac/spectral/spec_msg_proto.h
perf_pwr_offload/drivers/host/wlan/lmac_offload_if/ol_if_spectral.c
```

This section describes the host driver module for direct attach and offload architectures. The spectral host module is compiled as a separate kernel module. In order to effectively support both Generation-I and Generation-II hardware, the spectral host driver is divided into mainly two parts:

- A common module, which encapsulates the core spectral functionality independent of underlying hardware.
- A hardware dependent LMAC module, which has generation-specific functionality. That is, there is one implementation for Generation I Spectral, and another for Generation II.

The common module interacts with the hardware dependent LMAC module via an indirection table. The function pointer entries in the indirection table are populated at attach time with the respective entries. The spectral functions will only take a pointer to IC and spectral data structure; it avoids the use of SC data structure completely.



**Figure 15-6 Host Spectral Driver Architecture**

### 15.3.1.1 Spectral Indirection Table

#### Spectral Indirection Table Function Descriptions

The Spectral ops data structure holds pointers to hardware related functions. These are listed in [Table 15-10](#).

**Table 15-10 Spectral Indirection**

| Function              | Description                            |
|-----------------------|--|
| get_tsf64             | Get current TSF value                  |
| get_capability        | Get spectral capabilities              |
| set_rxfilter          | Set the RX filter values               |
| get_rxfilter          | Get the RX filter values               |
| is_spectral_active    | Check if Spectral is currently active  |
| is_spectral_enabled   | Check if Spectral is currently enabled |
| start_spectral_scan   | Start spectral scan                    |
| stop_spectral_scan    | Stop spectral scan                     |
| get_extension_channel | Get current Extension channel          |
| get_ctl_noisefloor    | Get Control channel noise floor value  |

**Table 15-10 Spectral Indirection (cont.)**

| Function              | Description                                  |
|-----------------------|--|
| get_ext_noisefloor    | Get Extension channel noise floor value      |
| configure_spectral    | Configure Spectral parameters                |
| get_spectral_config   | Get the current Spectral configurations      |
| get_ent_spectral_mask | Check if Enterprise mask is set for the Chip |
| get_mac_address       | Get MAC address of the device                |
| get_current_channel   | Get current operating channel                |
| reset_hw              | Reset hardware/chip                          |
| get_chain_noise_floor | Get noise floor on all chains                |

### 15.3.2 Spectral Indirection Table Mapping

Table 15-11 gives the mapping of Direct Attach LMAC and Offload functions to the Spectral Indirection Table.

**Table 15-11 Spectral Indirection Table Mapping**

| Function              | Direct Attach Mapping              | Offload Mapping                      |
|-----------------------|------------------------------------|--------------------------------------|
| get_tsf64             | ath_spectral_get_tsf64             | ol_if_spectral_get_tsf64             |
| get_capability        | ath_spectral_get_capability        | ol_if_spectral_get_capability        |
| set_rxfilter          | ath_spectral_set_rxfilter          | ol_if_spectral_set_rxfilter          |
| get_rxfilter          | ath_spectral_get_rxfilter          | ol_if_spectral_get_rxfilter          |
| is_spectral_active    | ath_spectral_is_spectral_active    | ol_if_is_spectral_active             |
| is_spectral_enabled   | ath_spectral_is_spectral_enabled   | ol_if_is_spectral_enabled            |
| start_spectral_scan   | ath_spectral_start_spectral_scan   | ol_if_start_spectral_scan            |
| stop_spectral_scan    | ath_spectral_stop_spectral_scan    | ol_if_stop_spectral_scan             |
| get_extension_channel | ath_spectral_get_extension_channel | ol_if_spectral_get_extension_channel |
| get_ctl_noisefloor    | ath_spectral_get_ctl_noisefloor    | ol_if_spectral_get_ctl_noisefloor    |
| get_ext_noisefloor    | ath_spectral_get_ext_noisefloor    | ol_if_spectral_get_ext_noisefloor    |
| configure_spectral    | ath_spectral_configure_params      | ol_if_spectral_configure_params      |
| get_spectral_config   | ath_spectral_get_params            | ol_if_spectral_get_params            |
| get_ent_spectral_mask | ath_spectral_get_ent_mask          | ol_if_spectral_get_ent_mask          |
| get_mac_address       | ath_spectral_get_mac_address       | ol_if_spectral_macaddr               |
| get_current_channel   | ath_spectral_get_current_channel   | ol_if_spectral_get_current_channel   |
| reset_hw              | ath_spectral_reset_hw              | ol_if_spectral_reset_hw              |
| get_chain_noise_floor | ath_spectral_get_chain_noise_floor | ol_if_spectral_get_chain_noise_floor |

### 15.3.2.1 Spectral Host Driver Initialization

The spectral module is initialized as a part of overall ATH layer initialization. The ATH layer initializes the spectral module. (Note that the ATH layer initialization routines are different for direct and offload architecture).

The spectral initialization takes care of memory allocation, gathering hardware spectral capabilities, initializing Netlink socket interface and configuring the default spectral scan parameters. It also populates the spectral indirection function table with dummy functions. The indirection table is later populated with relevant entries by the ATH layer. [Section 15.3.1.1](#) gives more information about the indirection table and its functionality.

### 15.3.2.2 Spectral IOCTL Interface

**NOTE** The spectral module provides the ath\_diag based IOCTLs listed in [Table 15-12](#) to configure and control spectral scan. The tool *sectratool* is provided to configure the spectral scan parameters using the below IOCTLs. Other applications too can programmatically use the same IOCTLs.

**Table 15-12 Spectral IOCTLs**

| IOCTL Command                | Description                                  |
|------------------------------|--|
| SPECTRAL_SET_CONFIG          | Configure spectral scan parameters           |
| SPECTRAL_GET_CONFIG          | Get current spectral scan parameters         |
| SPECTRAL_ENABLE_SCAN         | Enable spectral scan                         |
| SPECTRAL_DISABLE_SCAN        | Disable spectral scan                        |
| SPECTRAL_ACTIVATE_SCAN       | Start spectral scan                          |
| SPECTRAL_STOP_SCAN           | Stop spectral scan                           |
| SPECTRAL_SET_DEBUG_LEVEL     | Configure debug print level                  |
| SPECTRAL_IS_ACTIVE           | Checks if spectral scan is currently active  |
| SPECTRAL_IS_ENABLED          | Checks if spectral scan is currently enabled |
| SPECTRAL_GET_CAPABILITY_INFO | Gets the hardware spectral capability        |
| SPECTRAL_GET_DIAG_STATS      | Gets spectral stats (diagnostics)            |
| SPECTRAL_GET_CHAN_WIDTH      | Gets the current channel width               |

### 15.3.2.3 Spectral Scan Parameters

Spectral scan can be configured based on need. Various parameters can be configured using IOCTL SPECTRAL\_SET\_CONFIG. Refer to [Section 15.1.1](#) for the Generation I register descriptions, and to [Section](#) for Generation II register descriptions.

The SPECTRAL\_PARAMS\_T data structure is used. The design does not allow configuring individual spectral parameters, they can be only configured together in one shot.

Use the following methodology:

- Read current spectral configuration (Using SPECTRAL\_GET\_CONFIG)
- Modify relevant spectral parameters from the read value
- Configure the new parameters (Using SPECTRAL\_SET\_CONFIG)

```

typedef struct {
    u_int16_t ss_fft_period;
    u_int16_t ss_period;
    u_int16_t ss_count;
    u_int16_t ss_short_report;
    u_int8_t radar_bin_thresh_sel;
    u_int16_t ss_spectral_pri;
    u_int16_t ss_fft_size;
    u_int16_t ss_gc_ena;
    u_int16_t ss_restart_ena;
    u_int16_t ss_noise_floor_ref;
    u_int16_t ss_init_delay;
    u_int16_t ss_nb_tone_thr;
    u_int16_t ss_str_bin_thr;
    u_int16_t ss_wb_rpt_mode;
    u_int16_t ss_rssi_rpt_mode;
    u_int16_t ss_rssi_thr;
    u_int16_t ss_pwr_format;
    u_int16_t ss_rpt_mode;
    u_int16_t ss_bin_scale;
    u_int16_t ss_dBm_adj;
    u_int16_t ss_chn_mask;
    int8_t ss_nf_cal[AH_MAX_CHAINS * 2];
    int8_t ss_nf_pwr[AH_MAX_CHAINS * 2];
    int32_tss_nf_temp_data;
} SPECTRAL_PARAMS_T;

```

#### 15.3.2.4 Starting Spectral Scan

Spectral scan can be started via SPECTRAL\_ACTIVATE\_SCAN IOCTL. Spectral scan is started when spectral is both enabled and active. The latest spectral parameters are configured into the hardware and respective HAL related start functions are invoked (Generation I) or firmware commands sent (Generation II). While configuring the spectral parameters, the host spectral module also initializes a set of variables used in preparation of generating SAMP messages. This information includes number of FFT bins, FFT length and data length expected.

#### 15.3.2.5 Stopping Spectral Scan

The spectral scan can be stopped via SPECTRAL\_STOP\_SCAN IOCTL. In the case of Generation I, this will not affect the SPECTRAL\_ENABLE\_SCAN state.

#### 15.3.2.6 Processing Spectral Data

The Spectral data returned from the Generation I and Generation II hardware are different as explained in previous sections, hence there are two different handlers to handle the spectral data. Irrespective of the differences these data are finally transformed into the Spectral Analysis Messaging Protocol (SAMP) messages and sent to upper layer.

**NOTE** In-driver classification (under the SPECTRAL\_CLASSIFIER\_IN\_KERNEL flag) is no longer supported.

### 15.3.2.7 SAMP Message Params

The spectral module sends Spectral information to external modules using a custom protocol called Spectrum Analysis Message Protocol (SAMP). Data will be sent in big endian byte order. Some of the important members of the SAMP message structure are listed below:

```
typedef struct spectral_samp_msg {
    u_int32_t      signature;           /* Validates the SAMP message */
    u_int16_t      freq;                /* Operating frequency in MHz */
    u_int16_t      vhtop_ch_freq_seg1; /* VHT Segment 1 centre frequency
                                         in MHz */
    u_int16_t      vhtop_ch_freq_seg2; /* VHT Segment 2 centre frequency
                                         in MHz */
    u_int16_t      freq_loading;       /* How busy was the channel */
    u_int16_t      dcs_enabled;        /* Whether DCS is enabled */
    DCS_INT_TYPE   int_type;           /* Interference type indicated by
                                         DCS */
    u_int8_t       macaddr[6];          /* Indicates the device interface */
    SPECTRAL_SAMP_DATA samp_data;     /* SAMP Data */
} __ATTRIB_PACK SPECTRAL_SAMP_MSG;
```

SAMP Data contains the following members:

```
typedef struct spectral_samp_data {
    int16_t      spectral_data_len;    /* indicates the bin size */
    int16_t      spectral_data_len_sec80; /* indicates the bin size for secondary
                                         80 segment */
    int16_t      spectral_rssi;        /* indicates RSSI */
    int16_t      spectral_rssi_sec80; /* indicates RSSI for secondary 80
                                         segment */
    int8_t       spectral_combined_rssi; /* indicates combined RSSI from all
                                         antennas */
    int8_t       spectral_upper_rssi;   /* indicates RSSI of upper band */
    int8_t       spectral_lower_rssi;   /* indicates RSSI of lower band */
    int8_t       spectral_chain_ctl_rssi[MAX_SPECTRAL_CHAINS]; /* RSSI for control
                                         channel, for all antennas */
    int8_t       spectral_chain_ext_rssi[MAX_SPECTRAL_CHAINS]; /* RSSI for extension
                                         channel, for all antennas */
    u_int8_t     spectral_max_scale;   /* indicates scale factor */
    int16_t      spectral_bwinfo;      /* indicates bandwidth info */
    int32_t      spectral_tstamp;      /* indicates timestamp */
    int16_t      spectral_max_index;   /* indicates the index of max magnitude */
    int16_t      spectral_max_index_sec80; /* indicates the index of max magnitude
                                         for secondary 80 segment */
    int16_t      spectral_max_mag;     /* indicates the maximum magnitude */
    int16_t      spectral_max_mag_sec80; /* indicates the maximum magnitude for
                                         secondary 80 segment */
    u_int8_t     spectral_max_exp;     /* indicates the max exp */
    int32_t      spectral_last_tstamp; /* indicates the last time stamp */
    int16_t      spectral_upper_max_index; /* indicates the index of max mag in
                                         upper band */
} __ATTRIB_PACK SPECTRAL_SAMP_DATA;
```

```

    int16_t      spectral_lower_max_index; /* indicates the index of max mag in
lower band */
    u_int8_t     spectral_nb_upper;        /* Not Used */
    u_int8_t     spectral_nb_lower;       /* Not Used */
    SPECTRAL_CLASSIFIER_PARAMS classifier_params; /* indicates classifier
parameters */
    u_int16_t    bin_pwr_count;          /* indicates the number of FFT bins */
    u_int16_t    bin_pwr_count_sec80;    /* indicates the number of FFT bins in
secondary 80 segment */
    u_int8_t     bin_pwr[MAX_NUM_BINS];  /* contains FFT magnitudes */
    u_int8_t     bin_pwr_sec80[MAX_NUM_BINS]; /* contains FFT magnitudes for the
secondary 80 segment */
    struct INTERF_SRC_RSP interf_list;   /* list of interference sources */
    int16_t     noise_floor;            /* indicates the current noise floor */
    int16_t     noise_floor_sec80;      /* indicates the current noise floor for
secondary 80 segment */
    u_int32_t    ch_width;              /* Channel width 20/40/80 MHz */
} __ATTRIB_PACK SPECTRAL_SAMP_DATA;

```

### 15.3.3 Generation II Spectral Scan Firmware Functionality Overview

This section provides a brief overview of the spectral functionality offered by firmware, to provide context for the Host-Firmware interaction discussed in section [15.3.4](#).

The firmware provides three functionalities related to Spectral:

- Configuration of parameters requested by the host
- Starting/stopping spectral scan
- Delivery of Spectral PHY data errors for the duration spectral scan is active

The firmware has a Spectral WHAL module which carries out hardware register writes to configure host requested parameters, and also starts/stops spectral scan when triggered by the host. Refer to section [for a list of parameters](#). It should be noted that the module does not provide any means to read back the parameters, inline with WMI design practice. The offload LMAC spectral module takes care of tracking the values configured, and providing them to the user application/other host driver functions when requested via get calls.

When the WHAL module receives a configuration request from the host, it does not immediately write the corresponding value into the hardware register. It caches the value in an internal data structure. When a spectral scan is actually started, the requested value is written to hardware as part of the scan initialization sequence. When a spectral scan is started, the spectral-related PHY data errors are sent upwards to the host along the same receive path as other PHY data errors (radar, for example).

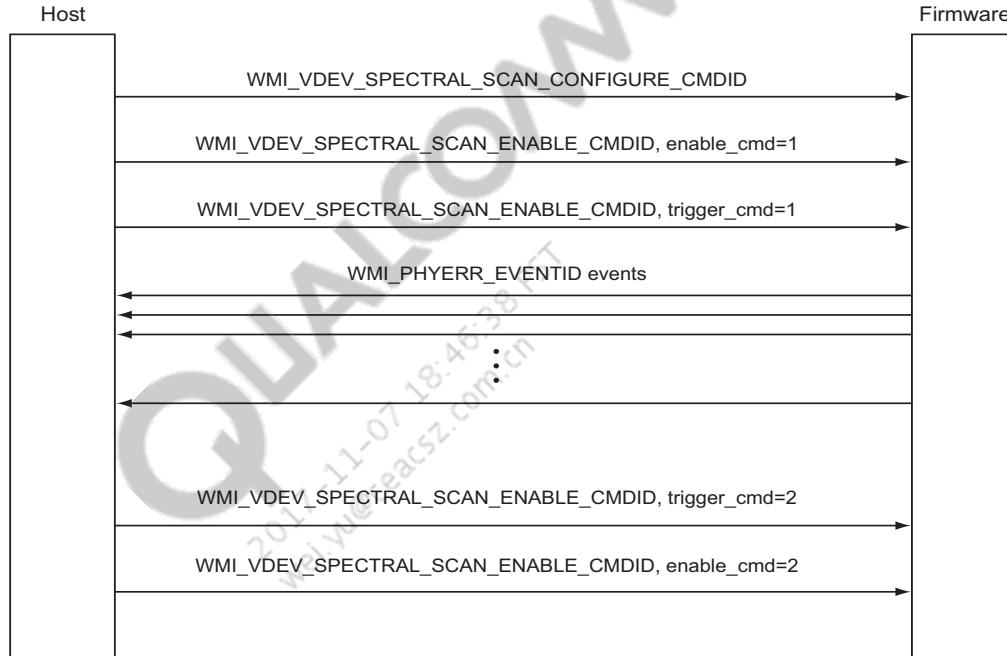
### 15.3.4 Generation II Spectral Scan Host-Firmware Interaction

#### Files

*perf\_pwr\_offload/drivers/include/wmi\_unified.h*  
*perf\_pwr\_offload/drivers/host/wlan/lmac\_offload\_if/ol\_if\_spectral.c*

Host-firmware interaction for Generation II spectral scan functionality happens through WMI. WMI commands are sent from the host to configure spectral. The spectral TLVs are sent from firmware to host inside WMI events. Note that ‘host’ in this context refers to the host driver. Other entities interacting with the host driver, such as spectral-based applications, could feed configuration information to it and be provided SAMP messages from it. The steps involved in configuration and TLV processing are given below.

**NOTE** Though the following commands are meant to act on a per VDEV basis, they are currently processed on a global (PDEV) basis. For now, the host simply sets the VDEV ID to that of the first VDEV on the radio.



**Figure 15-7 WMI Configuration and Event Sequencing for Spectral**

1. Configure the Spectral Parameters required, using the `WMI_VDEV_SPECTRAL_SCAN_CONFIGURE_CMDID` command. Refer to section for details about each parameter.

Cast each parameter to `A_UINT32`.

Refer to `drivers/wlan_modules/lmac/spectral/spectral.h` for defaults used for Spectral Classification (search for `SPECTRAL_SCAN_XXX_DEFAULT`; for example, for `SPECTRAL_SCAN_PERIOD_DEFAULT`, refer to #else of `ATH_SPECTRAL_USE_EMU_DEFAULTS`). Other applications areas built on top of spectral might require a different set of defaults in the future.

Note the following:

- Testing with `EXT_BLOCKER` PHY error code has not been carried out, since this scenario is not yet required for spectral classification.

- In keeping with WMI design recommendations, no equivalent read command has been implemented currently. The host offload LMAC driver keeps track of the parameters it has configured.
2. Enable spectral using the *WMI\_VDEV\_SPECTRAL\_SCAN\_ENABLE\_CMDID* command.  
*enable\_cmd* in the *wmi\_vdev\_spectral\_enable\_cmd* structure is set to 1 (enable).  
*trigger\_cmd* in the *wmi\_vdev\_spectral\_enable\_cmd* structure is set to 0 (ignore).
  3. Activate spectral using the *WMI\_VDEV\_SPECTRAL\_SCAN\_ENABLE\_CMDID*  
*trigger\_cmd* in the *wmi\_vdev\_spectral\_enable\_cmd* structure is set to 1 (trigger).  
*enable\_cmd* in the *wmi\_vdev\_spectral\_enable\_cmd* structure is set to 0 (ignore).
  4. Handle *WMI\_PHYERR\_EVENTID* events bearing spectral PHY error information. For more details, refer to section [15.3.4.1](#).
  5. Handle the requisite number of spectral samples, and proceed to steps [6](#) and [7](#) when done.
    - a. For an infinite spectral scan (*spectral\_scan\_count* in step 1 set to 0), the host driver can choose to wait for some dwell time period of its choice, all the while carrying out step [4](#) for each arriving event (this is the choice for use cases such as EACS with spectral scan). Alternatively, it can wait until spectral scan is stopped by the application (this is the choice for use cases such as spectral classification).
    - b. For a finite spectral scan (*spectral\_scan\_count* in step 1 set to some desired value, say 100), the requesting application has to keep track of the incoming spectral samples to determine when the count has been reached. No separate event is sent by the firmware to indicate exhaustion of the count. Once the application has received the desired number of samples, it should issue an ioctl to stop spectral scan which causes the host to proceed to steps [6](#) & [7](#).
  6. (Might become optional for finite spectral scans in the future, but required as at present)  
Deactivate Spectral using the *WMI\_VDEV\_SPECTRAL\_SCAN\_ENABLE\_CMDID*  
*trigger\_cmd* in the *wmi\_vdev\_spectral\_enable\_cmd* structure is set to 2 (clear trigger).  
*enable\_cmd* in the *wmi\_vdev\_spectral\_enable\_cmd* structure is set to 0 (ignore).
  7. Disable Spectral using the *WMI\_VDEV\_SPECTRAL\_SCAN\_ENABLE\_CMDID* command.  
*enable\_cmd* in the *wmi\_vdev\_spectral\_enable\_cmd* structure is set to 2 (disable).  
*trigger\_cmd* in the *wmi\_vdev\_spectral\_enable\_cmd* structure is set to 0 (ignore).

### 15.3.4.1 WMI\_PHYERR\_EVENTID processing details

#### Files

*perf\_pwr\_offload/drivers/include/wmi\_unified.h*  
*perf\_pwr\_offload/drivers/host/wlan/lmac\_offload\_if/ol\_if\_phyerr.c*  
*drivers/wlan\_modules/lmac/spectral/spectral\_phyerr.c*

1. In the host, the *WMI\_PHYERR\_EVENTID* event notification must be looped over to handle the (future) case where multiple *wmi\_single\_phyerr\_rx\_event* PHY notifications will be passed in an aggregate *wmi\_comb\_phyerr\_rx\_event* WMI PHY notification. Refer to *perf\_pwr\_offload/drivers/include/wmi\_unified.h* for the formats of the individual and aggregate

notifications. The looped handling of the WMI\_PHYERR\_EVENTID notifications is done in *ol\_ath\_phyerr\_rx\_event\_handler()*

2. In the loop, pull out the event header and data for each *wmi\_single\_phyerr\_rx\_event*.
  - a. Use *WMI\_UNIFIED\_PHYERRCODE\_GET()* on the event header to get the PHY error code.
  - b. If the code is *PHY\_ERROR\_SPECTRAL\_SCAN* (0x26 for AR9888, 0x04000000 for AR900B/QCA9984/QCA9888) or *PHY\_ERROR\_FALSE\_RADER\_EXT* (0x24 for AR9888, 0x01000000 for AR900B/QCA9984/QCA9888), the content is Spectral related; proceed to step c. Else, skip steps c and d.
  - c. Various information can be extracted from the event header using the following macros:
    - i. *WMI\_UNIFIED\_NF\_CHAIN\_GET()* for per-chain NF values
    - ii. *WMI\_UNIFIED\_RSSI\_COMB\_GET()* for combined RSSI
    - iii. *WMI\_UNIFIED\_RSSI\_CHAN\_GET()* for RSSI corresponding to *PRI20, SEC20, SEC40, SEC80*
  - d. The event data will contain Spectral TLVs. Process the TLVs. Refer to section [15.2.3.2](#) for more details on the formats of the TLVs.

## 15.4 Classification

This section provides details about the classifier algorithm. The FFT bins obtained during spectral scan are fed to the classifier module. These samples are analyzed for presence of interfering signals.

**NOTE** Spectral shares FFT engines with DFS. 8 additional FFT bins are reported for QCA9984/QCA9888, 4 on the left and 4 on the right of the band-edge. These have been introduced for DFS purposes, but they are reported for Spectral as well. The Spectral classification algorithms do not currently use these extra bins.

### 15.4.1 Microwave Oven Detection

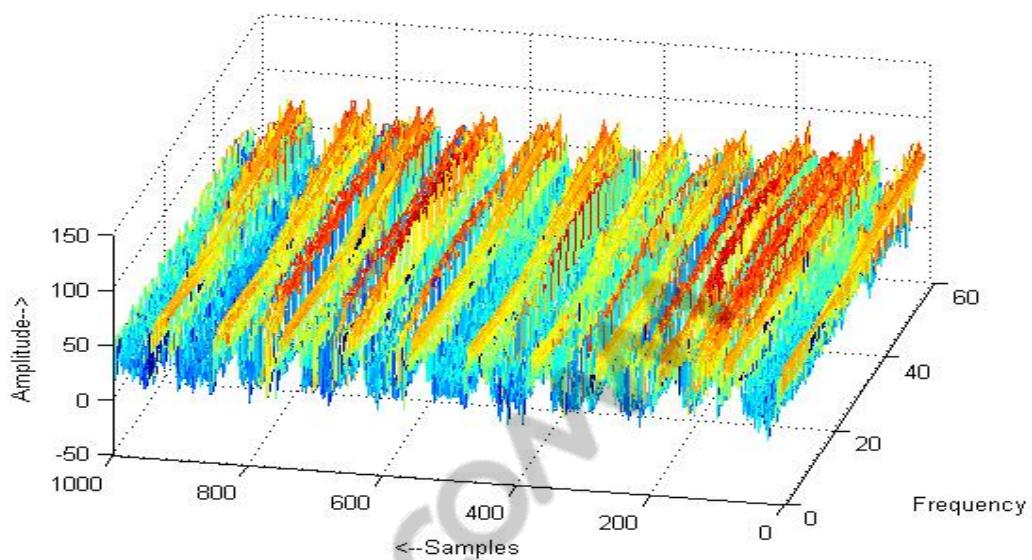
#### Function

*detect\_mwo*

#### Files

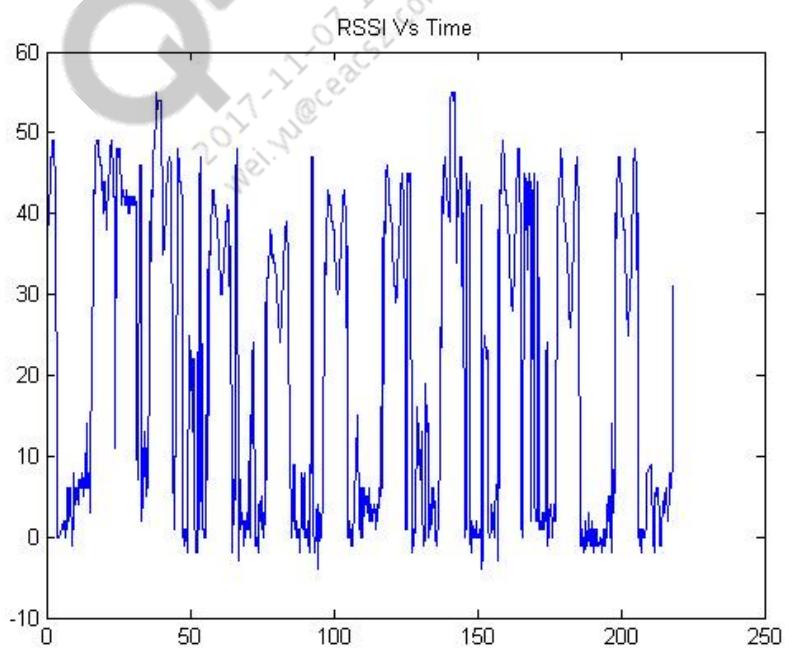
*apps/spectral/ath\_classifier.c*

Microwave oven (MWO) interference is present in channels 6 to 11. The interference is in a sweep pattern, from lower to higher with more energy seen in the higher bands. MWO interference as seen in channel 11 capture is plotted in [Figure 15-8](#). The bin energy increases towards the higher frequencies. Another characteristic of the MWO interference is that it is on for half of the power line period and absent in the other half (10 or 8.5 ms, depending on power line frequency of 50 or 60 Hz).



**Figure 15-8 MWO Interference Pattern In Channel 11**

Figure 15-9 shows the RSSI pattern of the spectral scan interference.



**Figure 15-9 RSSI Pattern of MWO Interference**

The MWO interference detection function makes use of both the spectral and temporal characteristics to do the detection. The detection logic is two folds: the basic detection, and repeat detection within a given time period to enhance reliability.

### 15.4.1.1 MWO Detection Logic

1. Check to see if the lower 10 bins has significantly less power than upper 10 bins. This condition should come at least once every MWO cycle.
2. Check if it has come within 8 ms. If so, this may be due to previous cycle.
3. Check if the detection came after 15 ms (greater than half the power cycle period). If so, this is the second cycle. Increment the MWO burst found counter.
4. If the number of bursts found in a given time is greater than threshold, declare that MWO interference has been found.
5. Check for at least two such detection in about 500 ms. If found, set the MWO interference flag to one.

**NOTE** This logic has been tested only with three brands of microwave ovens. It may require some modifications based on wider testing

### 15.4.1.2 Continuous Wave Interference Detection (for Field Devices)

#### Function

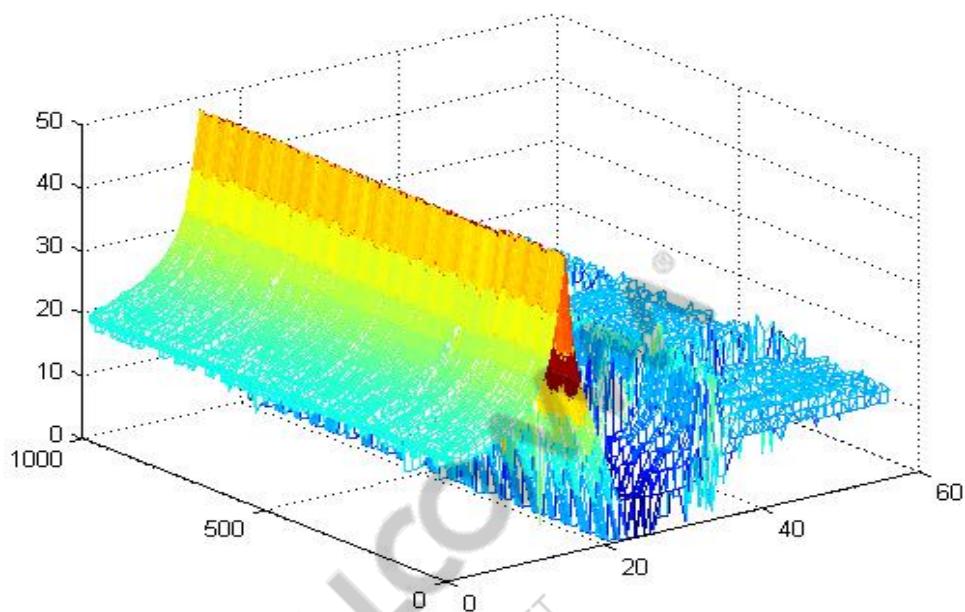
*detect\_cw*

#### File

*apps/spectral/ath\_classifier.c*

Continuous Wave (CW) Interference can be caused by Video Bridges (VB) and Baby Monitors (BM). They use frequency modulation technique to transmit signals in some of the ISM bands. They are persistent and completely block the WLAN channel in which they operate.

Figure 15-10 shows VB interference in channel 6, without any video signal connected. With the signal connected to VB, the signal moves from band to band.



**Figure 15-10 Video Bridge Interference in Channel 6**

#### 15.4.1.3 Continuous Wave Detection Logic

The detection logic for CW interference is essentially detection of sustained narrow band signal

1. Check if the RSSI is greater than threshold.
2. Find the peak bin.
3. Check if the combined power of peak bin and its adjacent bins is by a threshold more than three bins on either sides.
4. Check if the peak bin sum is greater than threshold. If so, a narrow band signal has been detected.
5. Check if this is the first time narrow band interference has been detected. If not, set a flag and start counting.
6. Each time a narrow band interference has been detected, increment the counter. If there are 30 detects within 10 ms, then it is confirmed as a narrow band signal detect.
7. If there is the narrow band signal goes missing for more than 1000 ms, reset the counter and detection flag.
8. Wait for at least two successful narrow band signal detections within 100 milliseconds before setting the CW interference flag.

In the case of 160/80+80 MHz scans, the FFT bins for each 80 MHz segment are passed through the above algorithm individually one after the other.

**NOTE** It is preferable to use field devices such as video bridges and baby monitors to test the above, rather than a theoretical test tone in the lab. Field devices tend to use various frequencies and at varying amplitudes. Thus there is a high probability that thresholds used in the algorithm implementation get crossed leading to successful detects, while at the same time reducing false positives. Also, a single tone at band centre may not be detected due to baseband filtering (the center is not used for WLAN Tx). If a tone generator has to be used, try to emulate usage of a few different frequencies and at varying amplitudes.

#### 15.4.1.4 WLAN (Wide Band) Signal Detection

##### Function

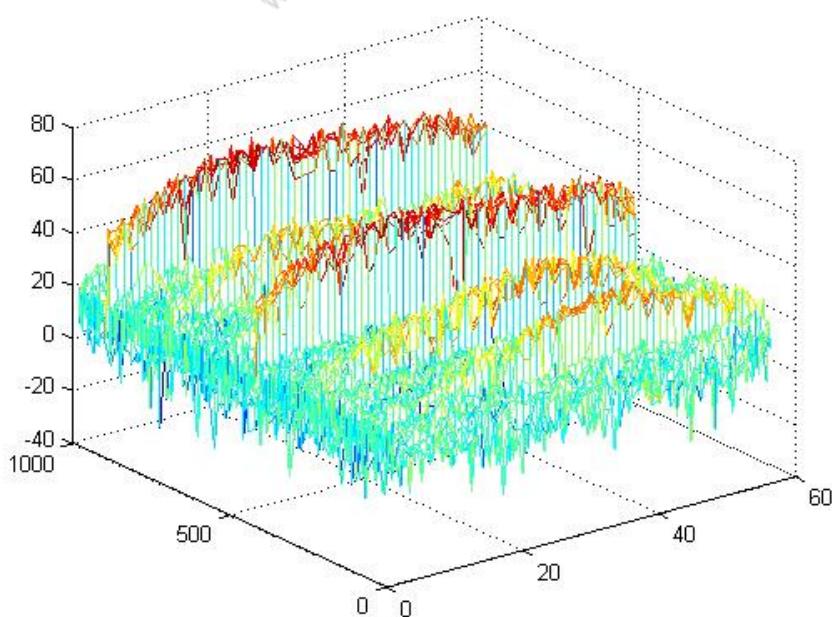
*detect\_wifi*

##### File

*apps/spectral/ath\_classifier.c*

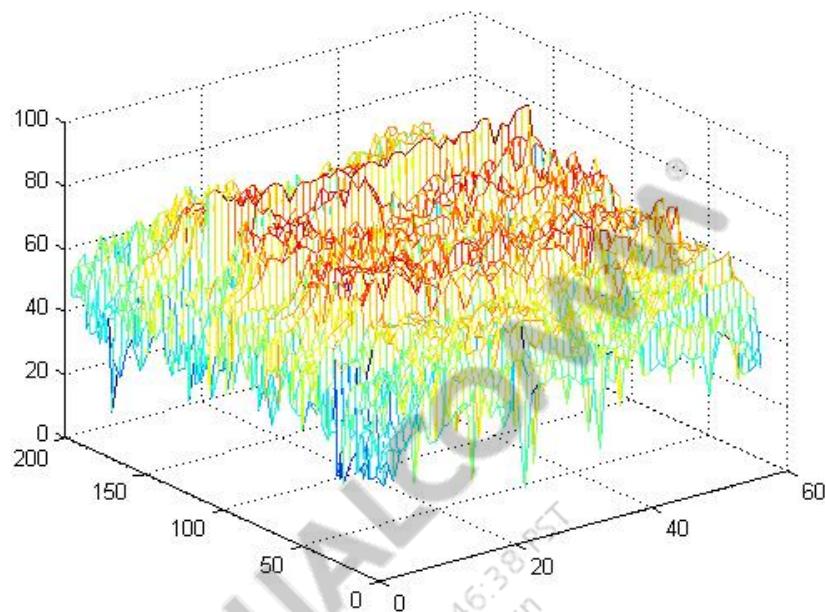
WLAN signals are wideband (WB) in that they occupy most of the bandwidth. However, they are usually stronger in the middle and weaker in the sides. This can be used as a way to detect WB signals. Another source of WB signals are Discrete Sequence Spread Spectrum (DSSS) cordless phones. It is difficult to differentiate the two, so the detection and classification applied is the same.

Figure 15-11 shows WLAN interference in channel 6. The center frequency bins are higher than band edges and it is a wideband signal.



**Figure 15-11 WLAN Interference in Channel 6**

Figure 15-12 shows DSSS cordless phone interference at 5.8 GHz (channel 161). The interference is too wideband in nature and hence difficult to differentiate from WLAN.



**Figure 15-12 DSSS Cordless Phone Interference in Channel 161**

#### 15.4.1.5 Wideband Detection Logic

1. Check if the RSSI is greater than threshold.
2. Find the peak bin value.
3. Check if at least half the bins have values within threshold based on peak bin value. If so, it is a wide band signal.
4. Check if the center bins are greater than the edge bins by a threshold value. If they are, then the basic detection is done.
5. Check if at least two wide band signals within 500 ms can be detected. If so, set the Wi-Fi interference flag.

The operating span is divided into chunks of 20 MHz and each chunk is searched separately. If no signal is found, then chunks of 40 MHz are used. And so on upwards.

In the case of 80+80 MHz scans, the FFT bins for each 80 MHz segment are passed through the above algorithm individually one after the other. In the case of 160 MHz scans, the FFT bins for the two segments are concatenated into a single bin set, and the above algorithm is run on this single set.

## 15.4.2 Frequency Hopping Spread Spectrum Detection

### Function

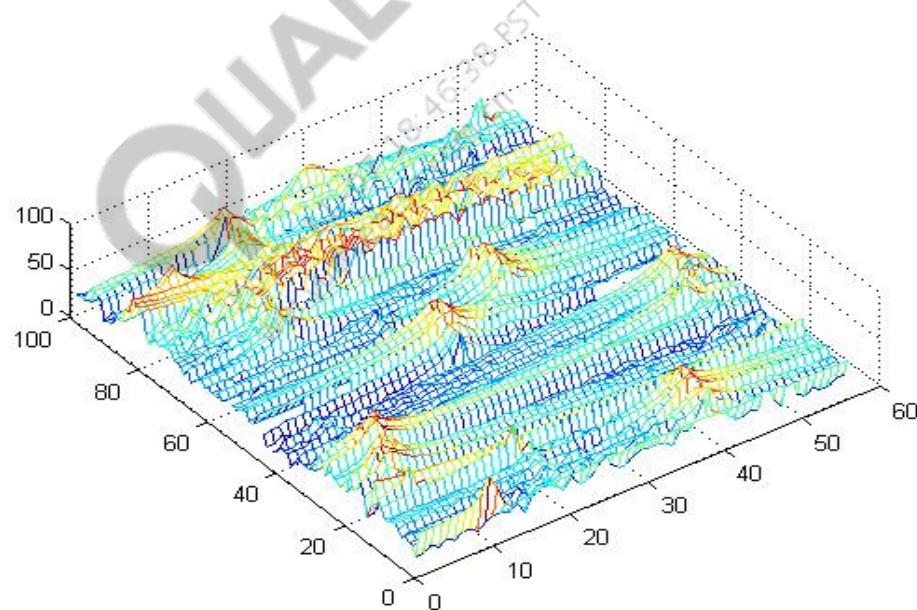
*detect\_fhss*

### File

*apps/spectral/ath\_classifier.c*

Frequency Hopping Spread Spectrum (FHSS) devices such as Bluetooth (BT) and FHSS cordless phones use a pre-assigned hopping sequence or random hopping sequence to dwell a short time in a given band and move on to the next band. BT uses the 1 MHz band and typical dwell time is 625  $\mu$ s or in multiples of the same. In the case of cordless phones, the sequence depends on the device, but typical times are within 5 ms.

Figure 15-13 shows FHSS cordless phone interference in channel 6 (with some WLAN interference). The peaks in this case persist for roughly 2 ms before hopping to a different channel.



**Figure 15-13 FHSS Cordless Phone Interference in Channel 6**

### 15.4.2.1 FHSS Detection Logic

1. Check if the RSSI is above a threshold, indicating reliable signal.
2. Check if the signal is narrow band by finding out the peak bin and checking whether the sum of peak and adjacent bins is greater than the sum of same number of bins to the right and left.
3. Get the bin number. Check if the bin number or the adjacent bin numbers are already recorded before. If they are, this is the same FHSS burst. If not, record the start time and the bin number.

4. If the bin number of the current burst does match the current burst, check when the burst started. If it is on for too long, it could be a CW interference, so reset the burst counter.
5. If the bin number does not match, save the difference between the start time and the stop time and record it and start a new burst.
6. If about 10 bursts have been collected, check the burst time of the second burst (the first one may have been captured half way). If at least 5 burst times are within a thresholds of the second burst time, this is most likely an FHSS burst.
7. If 2 such cases are detected within 5 second, mark FHSS interference as present.

In the case of 160/80+80 MHz scans, the FFT bins for each 80 MHz segment are passed through the above algorithm individually one after the other.

## 15.5 Spectraltool: Command Line Utility

### Files

*drivers/wlan\_modules/os/linux/spectraltool.c*

The command line utility *spectraltool* can be used to configure and query for the spectral scan parameters. Note that other applications might choose to configure the driver independent of *spectraltool* (using the same ioctls). Thus configurations written by *spectraltool* may be overwritten. Besides, some of the advanced settings, if altered from their defaults, might result in *athssd* (described in section ) not being able process the spectral data properly.

The important parameters available for configuration through *spectraltool* are given in [Table 15-13](#). For a more detailed description of the parameters below, refer to section . To read the current configuration, merely issue ‘*spectraltool -i wifiN*’.

**Table 15-13 Spectraltool Commands**

| Parameter    | Format  | Description  |
|--------------|---|--|
| fft_period   | <i>spectraltool -i wifiN</i><br><i>fft_period val</i>     | Set skip interval for FFT reports. (Not applicable for 11ac chipsets.)   |
| scan_period  | <i>spectraltool -i wifiN</i><br><i>scan_period val</i>    | Set spectral scan period. Period increment resolution is 256*Tclk, where Tclk = 1/44 MHz (Gmode), 1/40 MHz (Amode)   |
| scan_count   | <i>spectraltool -i wifiN</i><br><i>scan_count val</i>     | Set number of reports to return  |
| short_report | <i>spectraltool -i wifiN</i><br><i>short_report {1 0}</i> | Set to 1 to report only one set of FFT results per <i>spectral_scan_period</i> . (Not applicable for 11ac chipsets.)   |
| priority     | <i>spectraltool -i wifiN</i><br><i>priority {1 0}</i>     | Set priority.  |
| fft_size     | <i>spectraltool -i wifiN</i><br><i>fft_size val</i>       | Set the number of FFT data points to compute, defined as a log index:<br>num_fft_pts = 2 <sup>fft_size</sup><br>Value can range from 2 (num_fft_pts=4) to 9 (num_fft_pts=512).<br>(Only for 11ac chipsets) |

**Table 15-13 Spectraltool Commands (cont.)**

| Parameter       | Format   | Description   |   |  |   |  |   |  |   |   |
|-----------------|--|---|---|--|---|--|---|--|---|---|
| gc_ena          | spectraltool -i wifiN<br>gc_ena {1 0}  | Set to enable targeted gain change before starting the spectral scan FFT. (Only for 11ac chipsets)  |   |  |   |  |   |  |   |   |
| noise_floor_ref | spectraltool -i wifiN<br>noise_floor_ref val   | Set noise floor reference number (signed) for the calculation of bin power (dBm). (Only for 11ac chipsets)  |   |  |   |  |   |  |   |   |
| init_delay      | spectraltool -i wifiN<br>init_delay val  | Disallow spectral scan triggers after Tx/Rx packets by setting this delay value to roughly SIFS time period or greater. Delay timer counts in units of 0.25 $\mu$ s. (Only for 11ac chipsets)   |   |  |   |  |   |  |   |   |
| nb_tone_thr     | spectraltool -i wifiN<br>nb_tone_thr val   | Set number of strong bins (inclusive) per sub-channel, below which a signal is declared a narrow band tone. (Only for 11ac chipsets)  |   |  |   |  |   |  |   |   |
| str_bin_thr     | spectraltool -i wifiN<br>str_bin_thr val   | Set bin/max_bin ratio threshold over which a bin is declared strong, for spectral scan bandwidth analysis. (Only for 11ac chipsets)   |   |  |   |  |   |  |   |   |
| wb_rpt_mode     | spectraltool -i wifiN<br>wb_rpt_mode {1 0}   | Set this to 1 to report spectral scans as EXT_BLOCKER (phy_error=36), if none of the sub-channels are deemed narrow band. (Only for 11ac chipsets)  |   |  |   |  |   |  |   |   |
| rssi_thr        | spectraltool -i wifiN<br>rssi_thr val  | ADC RSSI must be greater than or equal to this threshold (signed Db) to ensure spectral scan reporting with normal PHY error codes (see rssi_rpt_mode in this table). (Only for 11ac chipsets)  |   |  |   |  |   |  |   |   |
| pwr_format      | spectraltool -i wifiN<br>pwr_format {0 1}  | Format of frequency bin magnitude for spectral scan triggered FFTs. (Only for 11ac chipsets) <table border="1" style="margin-left: 20px;"> <tr> <td>0</td><td>linear magnitude</td></tr> <tr> <td>1</td><td>log magnitude<br/>(<math>20 \cdot \log_{10}(\text{lin\_mag})</math>, 1/2 dB step size)</td></tr> </table>   | 0 | linear magnitude                       | 1 | log magnitude<br>( $20 \cdot \log_{10}(\text{lin\_mag})$ , 1/2 dB step size) |   |  |   |   |
| 0               | linear magnitude   |   |   |  |   |  |   |  |   |   |
| 1               | log magnitude<br>( $20 \cdot \log_{10}(\text{lin\_mag})$ , 1/2 dB step size)   |   |   |  |   |  |   |  |   |   |
| rpt_mode        | spectraltool -i wifiN<br>rpt_mode val  | Format of per-FFT reports to software for spectral scan triggered FFTs. (Only for 11ac chipsets) <table border="1" style="margin-left: 20px;"> <tr> <td>0</td><td>No FFT report (only pulse end summary)</td></tr> <tr> <td>1</td><td>2-dword summary of metrics for each completed FFT</td></tr> <tr> <td>2</td><td>2-dword summary + 1x-oversampled bins (in-band) per FFT. In the case of QCA9984/QCA988, 8 additional bins are reported, 4 bins to the left and right of the band-edge.</td></tr> <tr> <td>3</td><td>2-dword summary + 2x-oversampled bins (all) per FFT</td></tr> </table> | 0 | No FFT report (only pulse end summary) | 1 | 2-dword summary of metrics for each completed FFT                            | 2 | 2-dword summary + 1x-oversampled bins (in-band) per FFT. In the case of QCA9984/QCA988, 8 additional bins are reported, 4 bins to the left and right of the band-edge. | 3 | 2-dword summary + 2x-oversampled bins (all) per FFT |
| 0               | No FFT report (only pulse end summary)   |   |   |  |   |  |   |  |   |   |
| 1               | 2-dword summary of metrics for each completed FFT  |   |   |  |   |  |   |  |   |   |
| 2               | 2-dword summary + 1x-oversampled bins (in-band) per FFT. In the case of QCA9984/QCA988, 8 additional bins are reported, 4 bins to the left and right of the band-edge. |   |   |  |   |  |   |  |   |   |
| 3               | 2-dword summary + 2x-oversampled bins (all) per FFT  |   |   |  |   |  |   |  |   |   |
| bin_scale       | spectraltool -i wifiN<br>bin_scale val   | Number of LSBs to shift out to scale the FFT bins for spectral scan triggered FFTs. (Only for 11ac chipsets)  |   |  |   |  |   |  |   |   |
| dBm_adj         | spectraltool -i wifiN<br>dBm_adj {1 0}   | Set to 1 (with pwr_format=1), to report bin magnitudes converted to dBm power using the noisefloor calibration results. (Only for 11ac chipsets)  |   |  |   |  |   |  |   |   |
| chn_mask        | spectraltool -i wifiN<br>chn_mask val  | Set per chain enable mask to select input ADC for search FFT. (Only for 11ac chipsets)  |   |  |   |  |   |  |   |   |

Spectral FFT data can be dumped to a file by spectral tool using the following commands:

```
# spectraltool -i wifiX startscan
# spectraltool -i wifiX raw_data
# spectraltool -i wifiX stopscan
```

This captures 1000 samples of raw data and writes them into a file called ‘outFile’ in the current working directory. Each sample is presented in one line of output. In the case of 160/80+80 MHz,

there are two lines of output, one for each 80 MHz segment sample. The format of each line is as follows:

1st value: Sample sequence number in file. Will be the same for both 80 MHz segments, in the case of 160/80+80 MHz.

2nd value: Number of FFT bins (say ‘n’). Will be the same for both 80 MHz segments, in the case of 160/80+80 MHz.

Next n bytes are the FFT bin values.

The next value is the 64 bit timestamp at which the sample was captured. Will be the same for both 80 MHz segments, in the case of 160/80+80 MHz.

The value after that is the RSSI value. The last value is the Noise Floor.

### Interpretation of Spectral log as captured by *spectraltool*

|                |                |       |       |     |       |           |      |             |
|----------------|----------------|-------|-------|-----|-------|-----------|------|-------------|
| Sample Index 1 | Number of bins | Bin 1 | Bin 2 | ... | Bin N | Timestamp | RSSI | Noise floor |
| Sample Index 2 | Number of bins | Bin 1 | Bin 2 | ... | Bin N | Timestamp | RSSI | Noise floor |
| Sample Index N | Number of bins | Bin 1 | Bin 2 | ... | Bin N | Timestamp | RSSI | Noise floor |

The preceding diagram explains the spectral log as captured by the *spectraltool*.

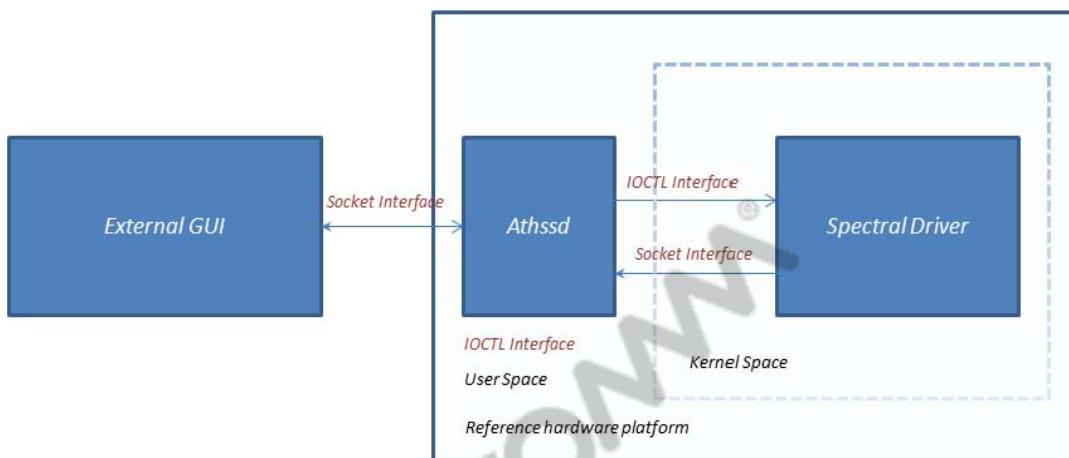
The individual fields are explained in detail in the following table.

|               |  |
|---------------|--|
| Sample Index  | Indicates the Nth Sample   |
| Number of bin | Indicates the number of bins, depends on the FFT size                |
| Bin (n)       | FFT Value at the Nth Bin   |
| Timestamp     | Indicates the time stamp (TSF) when the Spectral sample was received |
| RSSI          | Indicates the RSSI value   |
| Noise floor   | Indicates the observed Noise floor value                             |

## 15.6 Spectral and Classifier Demo Application: *athssd*

*athssd* is a demonstration command line application that uses the classification library. *athssd* feeds the spectral data gathered from the driver into the library and displays detected interferences and relevant statistics. It can also interface with an external GUI entity. Note that the GUI is not

typically provided to customers. The GUI is not currently tested for use with this release. The spectral analysis setup is shown in [Figure 15-14](#).



**Figure 15-14 athssd Spectral Analysis Setup**

## 15.6.1 Athssd: User Arguments

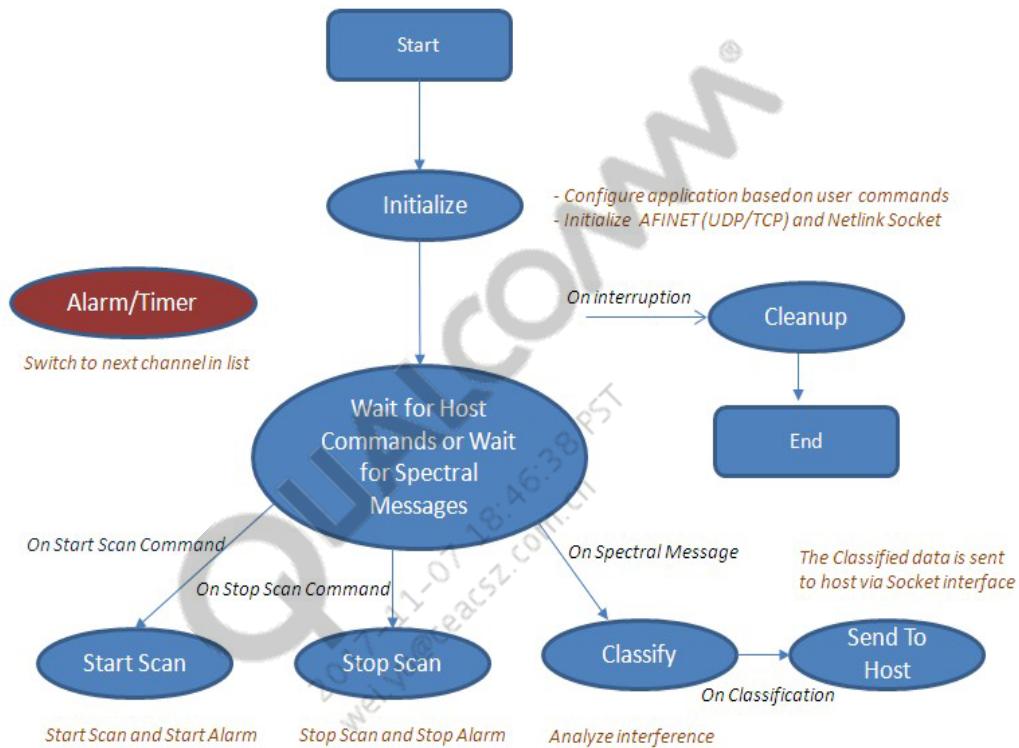
**Table 15-14 Athssd User Arguments**

| User argument    | Description   |
|------------------|---|
| -a               | Operate in 5 GHz band   |
| -d               | Enable debug prints   |
| -g               | Operate in 2.4 GHz band   |
| -h               | Print help message  |
| -i <radio_name>  | Indicates radio interface   |
| -j <device_name> | Indicates device interface  |
| -s <channel>     | Operate in standalone mode in provided channel number (0 indicates current channel) |
| -u               | Uses UDP socket   |
| -c               | Capture logs (0: None 1:MWO 2:CW 3:WiFi 4:FHSS 5:All)                               |

### 15.6.1.1 State diagram

This section describes the state transition diagram for the *athssd* application. The *athssd* application interfaces to the GUI program via a UDP socket interface. There is also support for TCP sockets. Interfacing with a GUI is optional. If there is no GUI, then *athssd* can be run in standalone mode (see argument for standalone mode in section [15.6.1](#)). The application uses Netlink interface to connect to the spectral driver. *athssd* translates the commands from the GUI into relevant spectral commands and configures the spectral driver via the IOCTL interface.

*athssd* receives the spectral data from the spectral driver in the Spectral Analysis Message Protocol (SAMP) format. This message is then passed to the classifier module. The classifier module will analyze the spectral data and detect if interference is present. Prints are output at the command line indicating presence/absence of interference, along with type information. Next, the SAMP message along with interference detection information is passed to the GUI (if connected) by *athssd*. The GUI will display the spectral information in the context of spectrogram and interference detection information. [Figure 15-15](#) shows the general *athssd* architecture.



**Figure 15-15 State transition diagram for *athssd***

## 15.7 Spectral debug enhancements

The following functional and debug enhancements are made to Spectral Analysis feature to enhance the usability and troubleshooting capability.

- Configurable number FFT reports logging using *spectraltool*—Number of FFT reports to be logged is made configurable (from fixed 1000 samples) between 1 to 50000.

This enables to capture FFT reports for 10 seconds assuming default duration of 200 microseconds.

Option `get_samples` has been enabled to capture samples between 1 to 50000 (inclusive).

Usage: `#spectraltool -i wifiX get_samples 1024`

- Enhance spectraltool to support low configuration system—Spectraltool used to fail on low configuration system, owing to the limitation on buffers for FFT reports. This limitation is now resolved, which causes the spectraltool to function in low configuration platforms.
- Enhance the logging formats for spectraltool—Spectraltool dumps the FFT reports in plain text format. Now option is added to dump into CSV format

Usage: #spectraltool -i wifiX get\_samples 1024 -l ','

- Extend Spectral debug to print Spectral PHY error report—Spectral debug level has been extended to dump just one Spectral PHY error report.

This enhancement improves debugging capabilities and enables verification of certain tests.

Usage: #spectraltool -i wifiX debug 4

- Implement timeout feature for athssd and spectraltool—Applications such as athssd and spectraltool are in closed loop while receiving the FFT reports from HW. If HW fails to generate FFT reports, then these applications are blocked and do not respond. To recover from this deadlock, timeout feature is implemented in athssd and spectraltool. If the FFT reports fail for any reason to arrive at application layer, these will timeout and give control to the user.

# 16 Band Steering Methods

---

This chapter describes the band steering approaches, such as Transmit Beamforming (TxBF), Single AP Band Steering, AP steering of legacy clients, band steering in QWRAP mode, Multi-AP Coordinated Steering and Adaptive Path Selection, and SSID Steering.

## 16.1 Transmit beamforming (TxBF)

This section describes the software for Transmit Beamforming (TxBF). It includes an introduction and a software flow.

To calculate an appropriate steering matrix for transmit spatial processing when transmitting to a specific beamformee, the beamformer needs to have an accurate estimate of the channel that it is transmitting over. There are two methods for BF: implicit feedback and explicit feedback. Only explicit feedback is supported.

The following is the list of supported TxBF capabilities:

| Capabilities   | Status                      |
|--|-----------------------------|
| Implicit TxBF Receiving Capable                      | Yes                         |
| Receive Staggered Sounding Capable                   | Yes                         |
| Transmit Staggered Sounding Capable                  | Yes                         |
| Receive NDP Capable                                  | No                          |
| Transmit NDP Capable                                 | No                          |
| Implicit TxBF Capable                                | Yes                         |
| Calibration (initiate and respond)                   | No                          |
| Explicit CSI TxBF Capable                            | No                          |
| Explicit Non-compressed Steering Capable             | Yes                         |
| Explicit Compressed Steering Capable                 | Yes                         |
| Explicit BF CSI Feedback (Delayed)                   | No                          |
| Explicit Non-compressed BF (Immediate only)          | Yes                         |
| Explicit Compressed BF (Immediate only)              | Yes                         |
| Minimal Grouping                                     | 3 for groups of 1, 2 and 4. |
| CSI No. of BF Antenna                                | N/A                         |
| Non-compressed steering number of Beamformer Antenna | 2 for 3 Tx antenna sounding |

| Capabilities                                     | Status                      |
|--|-----------------------------|
| Compressed steering number of Beamformer Antenna | 2 for 3 Tx antenna sounding |
| Channel Estimation Capability                    | 2 for 3 space time streams  |

## 16.1.1 Explicit feedback TxBF

When using explicit feedback, the beamformee makes a direct estimate of the channel from training symbols sent to the beamformee by the beamformer. The beamformee may prepare Channel State Information or Steering feedback (compressed or non compressed) based on an observation of these training symbols. The beamformee quantizes the feedback and sends it to the beamformer. The beamformer can use the feedback as the basis for determining transmit steering vectors.

**NOTE** Implicit beamforming (IBF) does not work with RDP319, whereas it works with AP152. This is an expected behavior.

### 16.1.1.1 Compressed or non-compressed BF Feedback

The software must initiate either compressed or non-compressed BF feedback. The hardware will handle the report in one frame. The software must collect the report more than one frame and pass it to hardware.

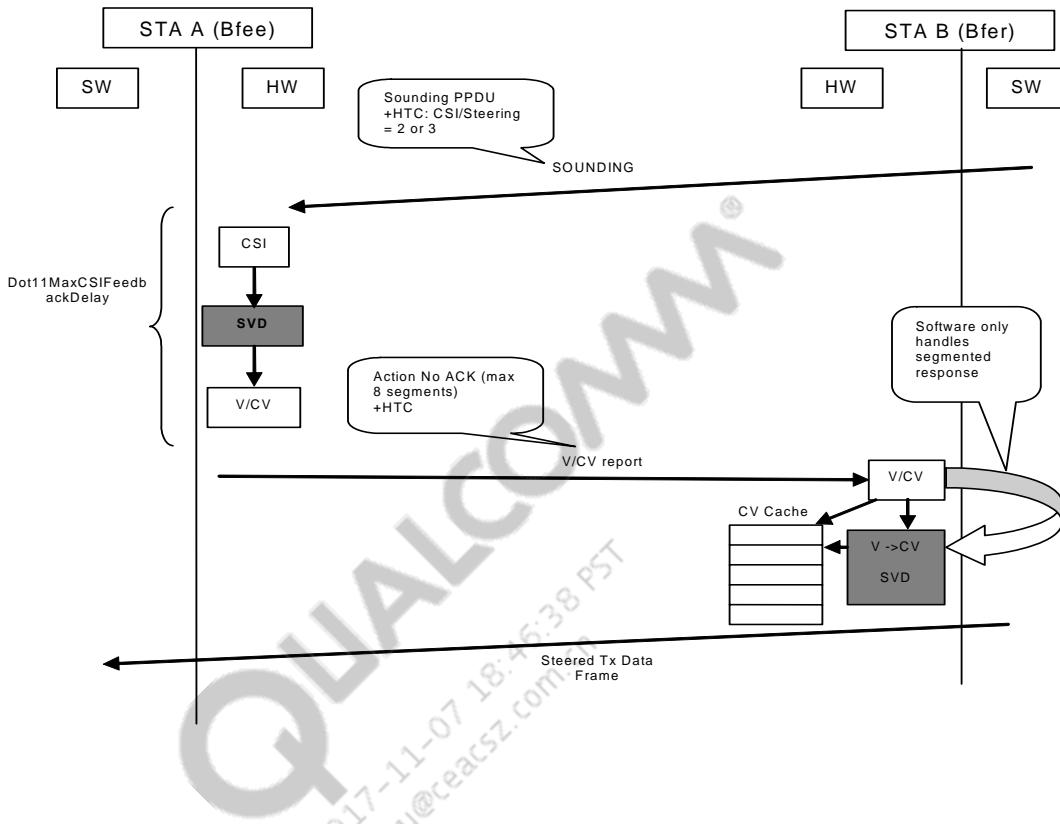


Figure 16-1 Explicit Feedback TxBF

### 16.1.1.2 HT capabilities subelement

To enable the beamforming function, the HT extend capability field and transmit beamforming capabilities field of HT capabilities sub-element should be set.

The format of HT capabilities sub-element is per the following table:

| Element ID | Length | HT Capabilities Info | A-MPDU Parameters | Supported MCS set | HT Extended Capabilities | Transmit Beamforming Capabilities | ASEL Capabilities |
|------------|--------|----------------------|-------------------|-------------------|--------------------------|-----------------------------------|-------------------|
| Octets:1   | 1      | 2                    | 1                 | 16                | 2                        | 4                                 | 1                 |

Bit 10 (+HTC support) of the HT Extended capabilities field should be enabled for BF protocol control. HT extended capabilities is shown in the following table.

| B0      | B1 B2               | B3 B7    | B8 B9        | B10          | B11          | B12 B15  |
|---------|---------------------|----------|--------------|--------------|--------------|----------|
| PCO     | PCO Transition Time | Reserved | MCS Feedback | +HTC Support | RD Responder | Reserved |
| Bits: 1 | 2                   | 5        | 2            | 1            | 1            | 4        |

The structure of the Transmit Beamforming Capabilities field is defined as follows:

| B0  | B1                                 | B2                                  | B3                  | B4                   | B5                                    | B6          | B7 | B8  |
|---|------------------------------------|-------------------------------------|---------------------|----------------------|---------------------------------------|-------------|----|---|
| Implicit Transmit Beamforming Receiving Capable | Receive Staggered Sounding Capable | Transmit Staggered Sounding Capable | Receive NDP Capable | Transmit NDP Capable | Implicit Transmit Beamforming Capable | Calibration |    | Explicit CSI Transmit Beamforming Capable |
| Bits:1  | 1                                  | 1                                   | 1                   | 1                    | 1                                     | 2           |    | 1   |

| B9                                       | B10                                  | B11 B12                                    | B13 B14  | B15 B16  | B17 B18          | B19 B20                                     |
|--|--------------------------------------|--|--|--|------------------|---|
| Explicit Non-compressed Steering Capable | Explicit Compressed Steering Capable | Explicit Transmit Beamforming CSI Feedback | Explicit Non-compressed Beamforming Feedback Capable | Explicit Compressed Beamforming Feedback Capable | Minimal Grouping | CSI Number of Beamformer Antennas Supported |
| Bits:1                                   | 1                                    | 2  | 2  | 2  | 2                | 2   |

| B21 B22   | B23 B24   | B25 B26                                     | B27 B28                       | B29 B31  |
|---|---|---|-------------------------------|----------|
| Non-compressed Steering Number of Beamformer Antennas Supported | Compressed Steering Number of Beamformer Antennas Supported | CSI Max Number of Rows Beamformer Supported | Channel Estimation Capability | Reserved |
| Bits:2  | 2   | 2   | 2                             | 3        |

Most of the aforementioned fields are set to indicate the capability of the specific function, or set to the number to indicate the number supported. Some special field bit definitions are as follows:

| Transmit Beamforming Capabilities field              | Definition  | Encoding   |
|--|---|--|
| Explicit Transmit Beamforming CSI Feedback           | Indicates whether this receiver can return CSI explicit feedback  | Set to 0 if not supported<br>Set to 1 for delayed feedback<br>Set to 2 for immediate feedback<br>Set to 3 for delayed and immediate feedback |
| Explicit Non-compressed Beamforming Feedback Capable | Indicates whether this receiver can return non-compressed beamforming feedback matrix explicit feedback.    | Set to 0 if not supported<br>Set to 1 for delayed feedback<br>Set to 2 for immediate feedback<br>Set to 3 for delayed and immediate feedback |
| Explicit Compressed Beamforming Feedback Capable     | Indicates whether or not this receiver can return compressed beamforming feedback matrix explicit feedback. | Set to 0 if not supported<br>Set to 1 for delayed feedback<br>Set to 2 for immediate feedback<br>Set to 3 for delayed and immediate feedback |

| Transmit Beamforming Capabilities field | Definition  | Encoding   |
|---|---|--|
| Minimal Grouping                        | Indicates the minimal grouping used for explicit feedback reports   | Set to 0 if the STA supports groups of 1 (no grouping)<br>Set to 1 indicates groups of 1, 2<br>Set to 2 indicates groups of 1, 4<br>Set to 3 indicates groups of 1, 2, 4 |
| Channel Estimation Capability           | Indicates the maximum number of space time streams (columns of the MIMO channel matrix) for which channel dimensions can be simultaneously estimated when receiving an NDP sounding PPDU or the extension portion of the HT long training fields in a staggered sounding PPDU | Set 0 for 1 space time stream<br>Set 1 for 2 space time streams<br>Set 2 for 3 space time streams<br>Set 3 for 4 space time streams                                      |

### 16.1.1.3 HT control field

Beamforming protocol is controlled by the HT control field, which is part of the frame header (4 octets wide) and is placed after QoS Control field.

The HT Control field is always present in a Control Wrapper frame, and is present in QoS data and management frames as determined by the order bit of the Frame Control field. The format of the 4-octet HT Control field is shown as follows:

| B0 B15                  | B16 B17              | B18 B19              | B20 B21  | B22 B23      | B24              | B25 B29  | B30           | B31             |
|-------------------------|----------------------|----------------------|----------|--------------|------------------|----------|---------------|-----------------|
| Link Adaptation Control | Calibration Position | Calibration Sequence | Reserved | CSI/Steering | NDP Announcement | Reserved | AC Constraint | RDG / More PPDU |
| Bits:16                 | 2                    | 2                    | 2        | 2            | 1                | 5        | 1             | 1               |

The link adaptation control field of first 16 bits is shown as follows:

| B0       | B1  | B2 B5 | B6 B8 | B9 B15   |
|----------|-----|-------|-------|----------|
| Reserved | TRQ | MAI   | MFSI  | MFB/ASEL |
| Bits:1   | 1   | 4     | 3     | 7        |

### 16.1.1.4 CSI/non-compressed beamforming and compressed beamforming action frame

Several Action frame formats are defined to support HT features. Three of them are for beamforming. The HT action field values are defined as follows:

|                    |                      |               |      |               |     |                            |                        |                                    |          |
|--------------------|----------------------|---------------|------|---------------|-----|----------------------------|------------------------|------------------------------------|----------|
| Action field value | 0                    | 1             | 2    | 3             | 4   | 5                          | 6                      | 7                                  | 8-255    |
| Meaning            | Notify Channel Width | SM Power Save | PSMP | Set PCO Phase | CSI | Non-compressed Beamforming | Compressed Beamforming | Antenna Selection Indices Feedback | Reserved |

All CSI/ Non-compressed beamforming / Compressed beamforming action frames are of an Action or an Action No Ack frame of category HT. The frame formats are similar. Refer to the IEEE 802.11n specification for details on MIMO control and each report field.

### CSI frame

|             |          |        |              |            |
|-------------|----------|--------|--------------|------------|
| Order       | 1        | 2      | 3            | 4          |
| Information | Category | Action | MIMO Control | CSI Report |

#### Non-compressed beamforming frame:

|             |          |        |              |                                   |
|-------------|----------|--------|--------------|-----------------------------------|
| Order       | 1        | 2      | 3            | 4                                 |
| Information | Category | Action | MIMO Control | Non-Compressed Beamforming Report |

#### Compressed beamforming frame

|             |          |        |              |                               |
|-------------|----------|--------|--------------|-------------------------------|
| Order       | 1        | 2      | 3            | 4                             |
| Information | Category | Action | MIMO Control | Compressed Beamforming Report |

### 16.1.1.5 Sounding PPDU

Sounding PPDUs are transmitted by STAs to enable the receiving STAs to estimate the channel between the transmitting STA and the receiving STA. The sounding PPDU should be set to 0 (Not Sounding) in the HT-SIG field. Send the proper extension HT\_LTF for training BF.

### 16.1.2 Hardware definitions for beamforming

This section describes the register definitions, Tx/Rx descriptor, and key cache for TxBF.

#### 16.1.2.1 Beamforming register definitions

|                   |       |         |         |        |
|-------------------|-------|---------|---------|--------|
| MAC_PCU_BASIC_SET | Group | MAC PCU | Address | 0x80A0 |
|-------------------|-------|---------|---------|--------|

| <b>bit#</b> | <b>name</b> | <b>type</b> | <b>reset</b> | <b>Description</b>   |
|-------------|-------------|-------------|--------------|--|
| 31:0        | MCS         | R/W         | 0x0          | BSSBasicMCSSet table of 802.11nD7.0.<br>It's directed mapping to the MCS of HT-SIG field and if set shown this MCS is supported.<br>That is,<br>bit0: MCS0<br>bit1: MCS1<br>bit31: MCS31 |

| <b>MAC_PCU_MGMT_SEQ</b> |             |             |              | <b>Group</b>  | <b>MAC PCU</b> | <b>Address</b> | <b>0x80A4</b> |
|-------------------------|-------------|-------------|--------------|---|----------------|----------------|---------------|
| <b>bit#</b>             | <b>name</b> | <b>type</b> | <b>reset</b> | <b>Description</b>  |                |                |               |
| 11:0                    | MIN         | R/W         | 0x0          | Minimum threshold of sequence number for management frame self-generated by hardware. |                |                |               |
| 15:12                   | Reserved    | R           | 0x0          | Reserved  |                |                |               |
| 27:16                   | MAX         | R/W         | 0x2FF        | Maximum threshold of sequence number for management frame self-generated by hardware. |                |                |               |
| 31:28                   | Reserved    | R           | 0x0          | Reserved  |                |                |               |

| <b>MAC_PCU_H_XFER_TIMEOUT</b> |                            |             |              | <b>Group</b>   | <b>MAC PCU</b> | <b>Address</b> | <b>0x831C</b> |
|-------------------------------|----------------------------|-------------|--------------|--|----------------|----------------|---------------|
| <b>bit#</b>                   | <b>name</b>                | <b>type</b> | <b>reset</b> | <b>Description</b>   |                |                |               |
| 6                             | extxbf_immediate_resp      | R/W         | 0x0          | Set to 1, indicates hardware to transfer immediate V/CV report in explicit TxBF  |                |                |               |
| 7                             | delay_extxbf_only_upload_h | R/W         | 0x0          | 0: Allow hardware transfer H/V/CV to software for delay explicit TxBF response<br>1: Only allow hardware transfer H to software for delay explicit TxBF response   |                |                |               |
| 8                             | extxbf_noack_norpt         | R/W         | 0x1          | This register is useful when register "extxbf_immediate_resp" is set to 1.<br>0: Allow hardware to response immediate report in explicit TxBF even the ack policy is no ack<br>1: If the ack policy of receiving sounding PPDU is no ack, hardware will transfer H/V/CV to software for delay response |                |                |               |

| TxBF_DBG |                  |      |       | Group  | Beamforming | Address | 0x10000 |
|----------|------------------|------|-------|--|-------------|---------|---------|
| bit#     | name             | type | reset | Description  |             |         |         |
| 1:0      | MODE             | R/W  | 0x0   | This is used for debugging by software.<br>If mode: 0x0, debug function will not active.<br>If mode: 0x1, software can force CV into CVCache<br>If mode: 0x2, software can force PV into H memory, and SVD+ will use it to apply power factor, then send it to BB. |             |         |         |
| 17:2     | CLIENT_TABLE     | R/W  | 0x0   | If mode: 0x2, software can force the certain client's CV in the CVCache, for example, if client_table is set to 0x1, it means that software forces CV into client1, and CVCache only outputs this client's CV to SVD+.   |             |         |         |
| 18       | SW_WR_V_DONE     | R/W  | 0x0   | In TXBF_DBG_MODE=0x2, software need to set this to 1 after software write all V into SVD V MEMORY  |             |         |         |
| 19       | DBG_IM           | R/W  | 0x0   | In TXBF_DBG_MODE=0x2, software need to set this to indicate svd_engine that PV in the SVD V MEMORY is for explicit or implicit<br>Set to 1 for implicit and 0 for explicit   |             |         |         |
| 20       | DBG_BW           | R/W  | 0x0   | In TXBF_DBG_MODE=0x2, software need to set this to indicate svd_engine that PV in the SVD V MEMORY is for BW20 or BW40<br>Set to 1 for BW40 and 0 for BW20   |             |         |         |
| 21       | CLK_CNTL         | R/W  | 0x0   | This is used to control svd_engine's clock.0x0 means that hardware control svd_engine's clock, otherwise svd_engine's clock always free-runs.  |             |         |         |
| 22       | REGULAR_SOUNDING | R/W  | 0x0   | Value: 0 for non-regular only, normal mode.<br>Value: 1 for regular only, debug mode.  |             |         |         |
| 23       | DBG_NO_WALSH     | R/W  | 0x0   | Value: 0 for wlash in debug mode.<br>Value: 1 for no walsh in debug mode.  |             |         |         |
| 24       | DBG_NO_CSD       | R/W  | 0x0   | Value: 0 for csd in debug mode.<br>Value: 1 for no csd in debug mode.  |             |         |         |
| 31:25    |                  | R    | 0x0   | Reserved   |             |         |         |

| TxBF |                      |      |       | Group  | Beamforming | Address | 0x10004 |
|------|----------------------|------|-------|--|-------------|---------|---------|
| bit# | name                 | type | reset | Description  |             |         |         |
| 1:0  | CB_TX                | R/W  | 0x0   | This is used for explicit beamforming. When explicit beamformee feedback CV reports to Beamformer, Beamformee's software can control 'codebook information' index to let SVD+ know how many bits need to be truncated.<br>Set to 0 for 1bit for psi, 3bits for phi<br>Set to 1 for 2bit for psi, 4bits for phi<br>Set to 2 for 3bit for psi, 5bits for phi<br>Set to 3 for 4bit for psi, 6bits for phi |             |         |         |
| 3:2  | NB_TX                | R/W  | 0x0   | This is used for explicit beamforming. When explicit Beamformee feedback V reports to Beamformer, Beamformee's software can control 'coefficient size' to let SVD+ know how many bits need to be truncated.<br>Set to 0 for 1bit for psi, 3bits for phi<br>Set to 1 for 2bit for psi, 4bits for phi<br>Set to 2 for 3bit for psi, 5bits for phi<br>Set to 3 for 4bit for psi, 6bits for phi            |             |         |         |
| 5:4  | NG_RPT_TX            | R/W  | 0x0   | This is used for explicit beamforming. When explicit Beamformee feedback V/CV reports to Beamformer, Beamformee's software can control 'grouping' to let SVD+ know how many carriers need to be grouped into one.<br>Set to 0 for Ng=1<br>Set to 1 for Ng=2<br>Set to 2 for Ng=4<br>The value 3 is reserved  |             |         |         |
| 7:6  | NG_CVCACHE           | R/W  | 0x0   | This is used for explicit/implicit beamforming. For explicit/implicit Beamformer's software can set this to control how many clients that CVCache can support.<br>Set to 0 for 4 clients be supported.<br>Set to 1 for 8 clients be supported.<br>Set to 2 for 16 clients be supported.<br>The value 3 is reserved   |             |         |         |
| 8    |                      | R    | 0x0   | Reserved   |             |         |         |
| 10:9 | TXCV_BFWEIGHT_METHOD | R/W  | 0x0   | This is used for calculating power factor.<br>0:no_weighting<br>1:max_power<br>2:keep_ratio  |             |         |         |
| 11   | RLR_EN               | R/W  | 0x0   | This is used for feedback V reports for Beamformee.<br>If set to 1, it means that SVD+ will the last row of V to be non-negative real numbers.   |             |         |         |
| 12   | RC_20_U_DONE         | R/W  | 0x0   | After writing radio coefficient into RC memory, software need to set this to 1 for BW20 upper  |             |         |         |
| 13   | RC_20_L_DONE         | R/W  | 0x0   | After writing radio coefficient into RC memory, software need to set this to 1 for BW20 lower  |             |         |         |

| TxBF  |            |      |       | Group   | Beamforming | Address | 0x10004 |
|-------|------------|------|-------|---|-------------|---------|---------|
| bit#  | name       | type | reset | Description   |             |         |         |
| 14    | RC_40_DONE | R/W  | 0x0   | After writing radio coefficient into RC memory, software need to set this to 1 for BW40 |             |         |         |
| 31:15 |            | R    | 0x0   | Reserved  |             |         |         |

| TXBF_TIMER |          |      |       | Group   | Beamforming | Address | 0x10008 |
|------------|----------|------|-------|---|-------------|---------|---------|
| bit#       | name     | type | reset | Description   |             |         |         |
| 4:0        | TIMEOUT  | R/W  | 0x0   | CVCache check the CV's lifetime every 1 ms. Every client has its own timestamp, when the timestamp - tsf_time > TIMEOUT, then this client's expire flag will be updated in the CVCache.<br>Set to 1 for 1 ms<br>Set to 2 for 2 ms<br>...<br>Set to 31 for 31 ms |             |         |         |
| 8:5        | ATIMEOUT | R/W  | 0x0   | When CVCache searches the CV, if hit and if timestamp - tsf_time > ATIMEOUT, then CVCache will notify MAC that this client is about to expire.<br>Set to 1 for 1 ms<br>Set to 2 for 2 ms<br>...<br>Set to 15 for 15 ms  |             |         |         |
| 31:9       |          | R    | 0x0   | Reserved  |             |         |         |

| TXBFSW | Group      | Beam-forming | Address | 0x1000C   |
|--------|------------|--------------|---------|---|
| bit#   | name       | type         | reset   | Description   |
| 0      | LRU_ACK    | RCLR         | 0x0     | After software triggers hardware to do LRU search, this bit will go high if hardware LRU search is done. Software then polls this bit to check if software LRU is done. Software reads/writes CVCACHE until this bit is high.<br>This bit is software-read-clear. |
| 9:1    | LRU_ADDR   | R            | 0x0     | After hardware does the LRU search for software, software must read LRU_ADDR when LRU_ACK is high. This address is used to tell the results of hardware LRU. This address points to each client's start-address.  |
| 10     |            | R            | 0x0     | Reserved  |
| 11     | LRU_EN     | R/W          | 0x0     | Software must set this bit high to indicate that hardware can do LRU search based on TXBF_SW_DEST_IDX.  |
| 18:12  | DEST_IDX   | R/W          | 0x0     | This is used with LRU_EN. It indicates to hardware to do LRU search based on this setting.  |
| 19     | LRU_WR_ACK | R/W          | 0x0     | ACK for CVCACHE write command.  |
| 20     | LRU_RD_ACK | R/W          | 0x0     | ACK for CVCACHE read command.   |

|       |                |     |     |  |
|-------|----------------|-----|-----|--|
| 21    | WALSH_CSD_MODE | R/W | 0x0 | Value: 0, keep track logic on WALSH for 3-stream regular sounding.<br>Value: 1, keep track logic on CSD for 3-stream regular sounding. |
| 31:22 |                | R   | 0x0 | Reserved.  |

| SVD_MEM 0 |           |      |       | Group   | Beamforming | Address | 0x11400~0x115C4 |
|-----------|-----------|------|-------|---|-------------|---------|-----------------|
| bit#      | name      | type | reset | Description   |             |         |                 |
| 31:0      | SVD_MEM_0 | R/W  | 0x0   | This is used for debugging by software. Software can directly read/write V into V memory.<br>Address: 0x11400 for tone 1<br>Address: 0x11404 for tone 2<br>...<br>Address: 0x115c4 for tone 114 |             |         |                 |

| SVD_MEM 1 |           |      |       | Group   | Beamforming | Address | 0x11600~0x117C4 |
|-----------|-----------|------|-------|---|-------------|---------|-----------------|
| bit#      | name      | type | reset | Description   |             |         |                 |
| 31:0      | SVD_MEM_1 | R/W  | 0x0   | This is used for debugging by software. Software can directly read/write V into V memory.<br>Address: 0x11600 for tone 1<br>Address: 0x11604 for tone 2<br>...<br>Address: 0x117c4 for tone 114 |             |         |                 |

| SVD_MEM 2 |           |      |       | Group   | Beamforming | Address | 0x11800~0x119C4 |
|-----------|-----------|------|-------|---|-------------|---------|-----------------|
| bit#      | name      | type | reset | Description   |             |         |                 |
| 31:0      | SVD_MEM_2 | R/W  | 0x0   | This is used for debugging by software. Software can directly read/write V into V memory.<br>Address: 0x11800 for tone 1<br>Address: 0x11804 for tone 2<br>...<br>Address: 0x119c4 for tone 114 |             |         |                 |

| SVD_MEM 3 |      |      |       | Group       | Beamforming | Address | 0x11a00~0x11bc4 |
|-----------|------|------|-------|-------------|-------------|---------|-----------------|
| bit#      | name | type | reset | Description |             |         |                 |

|      |           |     |     |   |
|------|-----------|-----|-----|---|
| 31:0 | SVD_MEM_3 | R/W | 0x0 | This is used for debugging by software. Software can directly read/write V into V memory.<br>Address: 0x11a00 for tone 1<br>Address: 0x11a04 for tone 2<br>...<br>Address: 0x11bc4 for tone 114 |
|------|-----------|-----|-----|---|

QUALCOMM®  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

| SVD_MEM_4 |           |      |       | Group   | Beamforming | Address | 0x11C00~0x11DC4 |
|-----------|-----------|------|-------|---|-------------|---------|-----------------|
| bit#      | name      | type | reset | Description   |             |         |                 |
| 31:0      | SVD_MEM_4 | R/W  | 0x0   | This is used for debugging by software. Software can directly read/write V into V memory.<br>Address: 0x11c00 for tone 1<br>Address: 0x11c04 for tone 2<br>...<br>Address: 0x11dc4 for tone 114 |             |         |                 |

| CVCACHE_MEM |         |      |       | Group  | Beamforming | Address | 0x12000~0x127FC |
|-------------|---------|------|-------|--|-------------|---------|-----------------|
| bit#        | name    | type | reset | Description  |             |         |                 |
| 31:0        | CVCACHE | R/W  | 0x0   | This is used for debugging by software. Software can directly read/write CV into CVCACHE.<br>Address: 0x12000 for CVCACHE addr0<br>Address: 0x12004 for CVCACHE addr1<br>...<br>Address: 0x127fc for CVCACHE addr511 |             |         |                 |

### Transmit Descriptor Control (TxC) for BF

| Word | Bit   | Name         | Description   |
|------|-------|--------------|---|
| 11   | 12    | vmf          | Virtual more fragment. If this bit is set, bursting is enabled for this frame.  |
| 11   | 28:25 | beam_form    | Transmit Beamforming in series 0~3. If this value is set, the current packet carries an array V before MPDU in current transmission series.<br>Bit 25 for transmission series 0.<br>Bit 26 for transmission series 1.<br>Bit 27 for transmission series 2.<br>Bit 28 for transmission series 3.                                 |
| 17   | 30    | calibrating  | Calibrating indication. If this bit is set then now is in radio calibration. This indication is used for notifying BB to apply correct MCSD to sounding PPDU, which is used to radio calibration. Use for cal pos=3.  |
| 19   | 28    | rts_htc_trq  | Sounding Request of RTS frame. Set to 1 to request the responder to transmit a sounding PPDU. When set to 0, the responder is not requested to transmit a sounding PPDU. This field is available when rts_enable is set to 1.   |
| 19   | 29    | not_sounding | Not Sounding field of HT-SIG. Set to 0 indicates that the PPDU is a Sounding PPDU. Set to 1 indicates that the PPDU is not a sounding PPDU. It's used for sending sounding PPDU in explicit feedback as a Beamformer. If rts_enable is set to 1 then this field just affects the RTS only; does not affect the next data frame. |

| Word | Bit   | Name   | Description   |
|------|-------|--------|---|
| 19   | 31:30 | Ness_0 | Ness (Number of Extension Spatial Streams) field of HT-SIG for transmission series 0.<br>Set to 0 indicates that no Extension HTLTF in PPDU<br>Set to 1 indicates that 1 Extension HTLTF in PPDU.<br>Set to 2 indicates that 2 Extension HTLTF in PPDU.<br>Set to 3 indicates that 3 Extension HTLTF in PPDU. |
| 20   | 31:30 | Ness_1 | Ness (Number of Extension Spatial Streams) field of HT-SIG for transmission series 1.<br>Set to 0 indicates that no Extension HTLTF in PPDU<br>Set to 1 indicates that 1 Extension HTLTF in PPDU.<br>Set to 2 indicates that 2 Extension HTLTF in PPDU.<br>Set to 3 indicates that 3 Extension HTLTF in PPDU. |
| 21   | 31:30 | Ness_2 | Ness (Number of Extension Spatial Streams) field of HT-SIG for transmission series 2.<br>Set to 0 indicates that no Extension HTLTF in PPDU<br>Set to 1 indicates that 1 Extension HTLTF in PPDU.<br>Set to 2 indicates that 2 Extension HTLTF in PPDU.<br>Set to 3 indicates that 3 Extension HTLTF in PPDU. |
| 22   | 31:30 | Ness_3 | Ness (Number of Extension Spatial Streams) field of HT-SIG for transmission series 3.<br>Set to 0 indicates that no Extension HTLTF in PPDU<br>Set to 1 indicates that 1 Extension HTLTF in PPDU.<br>Set to 2 indicates that 2 Extension HTLTF in PPDU.<br>Set to 3 indicates that 3 Extension HTLTF in PPDU. |

### Transmit Descriptor Status (TxS) for BF

| Word | Bit | Name              | Description   |
|------|-----|-------------------|---|
| 8    | 18  | txbf_bw_mismatch  | Bandwidth mismatch indication for TxBF. If set, shows that the bandwidth of CV data is not same as the bandwidth of transmitting PPDU, then hardware will send the PPDU out without beamforming.                        |
| 8    | 19  | txbf_stream_miss  | Stream miss indication for TxBF. When set, indicates that the CV information in CV cache is not enough for transmitting steered PPDU with current Tx rate, but is still transmitting this PPDU out without beamforming. |
| 8    | 23  | txbf_dest_miss    | Destination miss indication for TxBF. When set, indicates there is no CV for this destination. The PPDU is transmitted out without beamforming.   |
| 8    | 24  | txbf_expired_miss | Time expired indication for TxBF. When set, indicates two kinds of status:<br>1. The left-time of CV for this transmission destination is lower than the threshold set by software.<br>2. CV is expired.                |

## Receive Descriptor Status (RxS) for BF

| Word | Bit   | Name                 | Description  |
|------|-------|----------------------|--|
| 2    | 12    | More                 | More descriptors in this frame flag. If set, then this is not the final descriptor of the frame. If clear, then this descriptor is the final one of the frame. Valid for all descriptors.  |
| 2    | 22    | hw_upload_data       | Indicates the data carried by current descriptor is that hardware upload for TxBF using. It could be H, V or CV data. The upload data is valid only when the field "hw_upload_data_valid" at RXS 4 bit [7] is set. See the field of "hw_upload_data_type" at RXS 11 bit [26:25] for data type uploaded. Valid for all descriptors. |
| 4    | 4     | not_sounding         | Receive packet not sounding flag. If this value is clear, then the receive frame is a sounding PPDU. If this value is set, the receive frame is not a sounding PPDU.   |
| 4    | 6:5   | Ness                 | Receive packet Ness (Number of Extension Spatial Streams) field. This value shows the number of Extension Spatial Streams per receiving frame.   |
| 4    | 7     | hw_upload_data_valid | Specifies whether the contents of the hardware upload data are valid   |
| 11   | 26:25 | hw_upload_data_type  | Indicate the hardware upload data is H, V or CV. The upload data is valid only when the field "hw_upload_data_valid" at RXS 4 bit [7] is set.<br>01: Upload is H<br>10: Upload is V<br>11: Upload is CV  |

### Key cache for BF

The key cache must be updated after association according to the remote capability of TxBF. The addresses that need to be updated are stagger\_sounding\_cap[48] CEC[46:45], MMSS[44:42] of the key cache third address.

## 16.1.3 Beamforming software flow

The software flow for BF can be divided into the following: initialization, after new association, Tx rate adaption, Tx descriptor setting, Tx descriptor status handling, Rx descriptor handling, Rx frame parsing, and CV update mechanism.

### 16.1.3.1 Initialization

At initialization, the software initializes the TxBF related register first when HAL is attached. This is done by the ar9300\_init\_txbf routine in ar9300\_txbf.c. It should then initialize TxBF capability for setting the information element. This is done in the ar9300\_fill\_txbf\_capabilities routine in ar9300\_txbf.c. These routines will be also included during software reset.

### 16.1.3.2 After new association

After a new association, the software will exchange the information element. According to remote and internal capability, the type of TxBF that should be used in this node will be determined. The routine is done in ieee80211\_match\_txbfcapability of ieee80211\_txbf.c.

If one type of BF is used at this node, it should initialize the key cache for the TxBF used. The routine is in ieee80211\_set\_TxBF\_keycache of ieee80211\_txbf.c. A timer for software CV time out used at this stage should be initialized.

### 16.1.3.3 Tx rate adaptation

Because gain is improved if TxBF is applied, the valid rate table is different compared to when TxBF was disabled. A 7-bit field “validTxBF” is used in the rate table to indicate whether or not the rate in rate table should be used for the current association.

These 7 bit value are

- Bit0: indicator this rate should be used at 3 stream and NESS = 2.
- Bit1: indicator this rate should be used at 2 stream and NESS = 2.
- Bit2: indicator this rate should be used at 3 stream and NESS = 1.
- Bit3: indicator this rate should be used at 2 stream and NESS = 1.
- Bit4: indicator this rate should be used at one stream and NESS = 1.
- Bit5: indicator this rate should be used at 20 MHz bandwidth
- Bit6: indicator this rate should be used at 40 MHz bandwidth.

After information element is exchanged, software sets the valid rate table for the rate adaptation used. If this node uses TxBF, a rate according to “validTxBF” field and the capability of current node should be set up.

Because the software does not apply TxBF at the sounding frame, the performance is worse than a normal frame at this time. The rate used at the sounding frame must be lower than a normal frame. The software must also set a sounding swap table if TxBF is required at this node.

The sounding swap table is a one dimension table. The value in the table is swapped rate code. The software should use the current MCS value as an index to extract the swap rate code. Because the performance degradation may be different at various bands and bandwidths, there are four tables used for sounding swapping (“sounding\_swap\_table\_A\_40”, “sounding\_swap\_table\_A\_20”, “sounding\_swap\_table\_G\_40”, “sounding\_swap\_table\_G\_20”). These tables are in ratectrl\_11n.c.

At the rate control stage of each frame, after it picks up rate for current frame, the software should decide according to current rate whether a steering vector should be applied (by using a macro “VALID\_TXBFRATE(\_rate, \_nss)”). For the sounding frame, the software should swap the rate using the sound rate swap table. It should also disable short GI and STBC according to current settings.

After Tx is complete, the status for statistics should be updated. If the current frame is the sounding frame, it cannot be taken into statistics.

#### 16.1.3.4 Tx descriptor setting

While preparing a Tx data packet, the software checks if this frame must be steered to set the beam\_form of the Tx descriptor control (TXC word 11 bit 25:28) in the “set\_11n\_tx\_bf\_flags(\_series, \_index)” macro. If it is a sounding frame, the software clears “not\_sounding” and set NESS according to the current rate. This is done by the routine “ar9300\_set\_11n\_txbf\_sounding” in ar9300\_txbf.c.

#### 16.1.3.5 Tx descriptor status handler

For TxBF, the Tx descriptor handler checks the hardware report status for TxBF. If the status not zero, it should initial a trigger to start next C/V update. This is done in “ieee80211\_release\_wbuf\_internal” routine of ieee80211\_output.c.

#### 16.1.3.6 Rx descriptor status handler /Rx frame parsing

For TxBF, the Rx descriptor handler should check the “more” bit (RXS word 2 bit 13) first. If this bit is set, it means that the more descriptor is waiting for parsing. Then it should check “hw\_upload\_data\_valid” (RXS word 4 bit 7). If this bit is set, it indicates that the next descriptor is for hardware upload data. It should parse the current frame to determine what kind of data is in the next descriptor, and send V or CV report according to the current frame content. The related code is in “ath\_rx\_bf\_process” of ath\_recv.c and “ath\_rx\_bf\_handler” of ath\_edma\_recv.c.

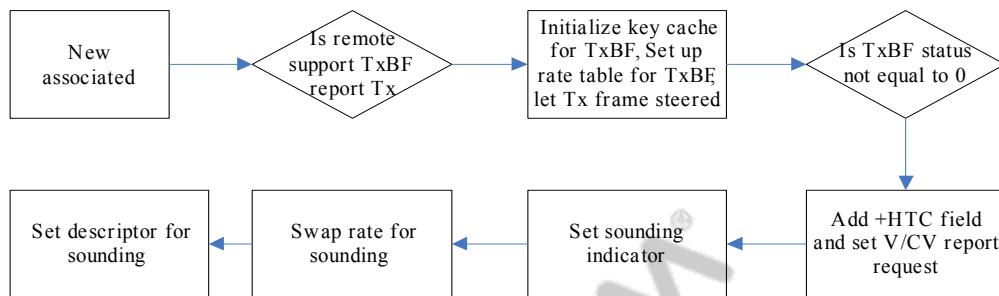
#### 16.1.3.7 CV update mechanism

The mechanism uses a software timer to initial CV update. The algorithm is as follows, and related codes are “ieee80211\_release\_wbuf\_internal” of ieee80211\_output.c.

When there is TxBF traffic, it will trigger a TxBF status report. If the status report is not zero, it will trigger a software timer. When the timer is expired, a sounding frame will be sent to request a CV update and set another short timer for CV received. If the CV report is not received before the short timer expires, the CV report may be lost. Another sounding frame is sent to request a CV update.

#### 16.1.3.8 Software flow for explicit TxBF

The following software flow is for explicit TxBF only:



1. When newly associated, the software will check the remote information element to see if it supports a compressed or non-compressed TxBF report.
2. If there is remote support, the local setting for support with TxBF compressed or non-compressed request is enabled. Then it will set the indicator (ni->ni\_explicit\_compbpf, ni->ni\_explicit\_noncompbpf, ni->ni\_implicit\_bf) to enable the TxBF function.
3. After the indicator has been set, it will start to set keycache for TxBF (“ieee80211\_set\_TxBF\_keycache” in ieee80211\_txbf.c). Then the software starts to initialize the rate table and sounding swap rate table for TxBF use.
4. For any frame transmission, if the current Tx rate can be steered, the software will set “steered” at Tx descriptor to start TxBF Tx.
5. If the TxBF status report is zero, it means that the CV cache values are good, and no further updates are needed. Otherwise, a sounding frame should be sent to initialize a C/V update.
6. If a C/V update is required, it should add +HTC field and send a V/CV request when a 80211 frame header is encapsulated. The, mark this frame as a sounding frame. Do not form an aggregation frame to transmit data.
7. When the frame is a sounding frame, a sounding rate table should be used to swap the rate index with the original one when a rate is chosen.
8. The sounding related field should be set in the Tx descriptor to transmit sounding.

A loop is formed between <4> ~<8> to process the TxBF transmission.

For the beamformee’s part, if a delayed report is required, Rx processes more bits to send a V/CV report back. An immediate report is done by hardware. No software effort is required for an immediately report.

#### 16.1.4 Enable TxBF

Use iwpriv to set “TxBFCTL” to enable TxBF control. Example:

```
iwpriv wifi0 TxBFCTL #value
```

The definition of the value is as follows:

bit0: reserved

- bit1: enable Explicit Beamforming with non-compressed report request (Beamformer side)
- bit2: enable Explicit Beamforming with compressed report request (Beamformer side)
- bit3: reserved
- bit4: enable Explicit Beamforming with non-compressed immediate report (Beamformee side)
- bit5: enable Explicit Beamforming with compressed immediate report (Beamformee side)
- bit6: enable Explicit Beamforming with non-compressed delayed report (Beamformee side)
- bit7: enable Explicit Beamforming with compressed delayed report (Beamformee side)

The default value for AP side is 246 (TxBF enabled).

The software can also use “GetTxBFCT” to read current settings. If `iwpriv wifi0 GetTxBFCTL` returns an error, the code in the current platform does not enable TxBF. The firmware should be changed.

### 16.1.5 Change CV update period

Use `iwpriv` to set “`SWCvtimOut`” to set CV cache update period. Example:

```
iwpriv wifi0 set_swcvtimout #value
```

The default value for AP side is 1000. This value is in ms.

“`get_cvtimeout`” can be used to read current settings. If `iwpriv wifi0 get_cvtimeout` returns an error, the code in the current platform does not enable TxBF. The firmware should be changed.

The hardware settings will take effect immediately (delayed report/immediate report swap). Other settings will take effect after next association.

## 16.2 Single AP band steering

The Wi-Fi Self-Organizing Network (Wi-Fi SON) band steering brings some new steering mechanisms and algorithms in the Load Balancing Daemon (lbd) to handle more scenarios and make use of features supported on newer Wi-Fi devices.

[Table 16-1](#) describes a brief summary of the changes in band steering between the initial (non-Wi-Fi SON) implementation and Wi-Fi SON implementation, with more details provided in subsequent sections.

**Table 16-1 Feature comparison between phases**

| Feature                               | Initial Band Steering Implementation  | Wi-Fi SON Implementation  |
|---------------------------------------|---|---|
| Pre-association steering              | <ul style="list-style-type: none"> <li>■ Only when overloaded</li> <li>■ For steering to 2.4 GHz, permitted regardless of RSSI.</li> <li>■ For steering to 5 GHz, requires uplink RSSI to be greater than the RSSISteeringPoint threshold in the WlanIF5G_Adv section of the configuration.</li> </ul>  | <ul style="list-style-type: none"> <li>■ Only when overloaded</li> <li>■ For both bands, requires RSSI to meet a minimum threshold.</li> </ul>  |
| Idle post-association steering        | <ul style="list-style-type: none"> <li>■ Legacy device mechanism only (blacklists + probe response withholding + forced disassociation)</li> <li>■ RSSI on non-serving band estimated with fixed offsets.</li> <li>■ For upgrade steering (2.4 GHz -&gt; 5 GHz), blacklist is removed once steering completes.</li> <li>■ For downgrade steering (5 GHz -&gt; 2.4 GHz), blacklist remains in place for a configurable amount of time (1 day by default).</li> <li>■ Default thresholds allow for upgrade and downgrade steering based on RSSI and overload conditions.</li> </ul> | <ul style="list-style-type: none"> <li>■ 802.11v BSS Transition Management (BTM) for devices that advertise support and behave well with it</li> <li>■ Legacy device mechanism (same as phase 1) for non-802.11v BTM capable devices (as well as those that are characterized as not behaving well with it)</li> <li>■ RSSI on non-serving band estimated with fixed offsets adjusted by any negative delta in the transmission power.</li> <li>■ For upgrade steering (2.4 GHz -&gt; 5 GHz), blacklist is removed once steering completes.</li> <li>■ For downgrade steering (5 GHz -&gt; 2.4 GHz), blacklist remains in place only for 15 minutes, so long as the channel is not overloaded.</li> <li>■ Default thresholds only target upgrade steering based on RSSI (as clients generally handle downgrade fairly well).</li> </ul> |
| Active post-association steering      | Not supported   | <ul style="list-style-type: none"> <li>■ Supported for clients that support 802.11k and 802.11v that are deemed active steering friendly</li> <li>■ Triggers use uplink RSSI and downlink last MCS</li> </ul>   |
| Interference Avoidance Steering (IAS) | Not supported   | <ul style="list-style-type: none"> <li>■ Detects STAs being impacted by interference hidden from the AP and steers them to a different channel</li> </ul>   |
| Steering prohibit time (hysteresis)   | 5 minutes for all devices   | <ul style="list-style-type: none"> <li>■ 30 seconds when 802.11v BSS Transition Management is used</li> <li>■ 5 minutes for legacy steering mechanism</li> </ul>  |
| Steering unfriendliness               | <ul style="list-style-type: none"> <li>■ Fixed duration of 10 minutes (configurable)</li> <li>■ Clients considered unfriendly will not be steered in any manner until the timer expires</li> </ul>  | <ul style="list-style-type: none"> <li>■ Exponential backoff scheme with a starting value of 10 minutes and a maximum value of 1 week (both configurable)</li> <li>■ Clients considered unfriendly will not be steered in any manner until the timer expires</li> </ul>   |

**Table 16-1 Feature comparison between phases (cont.)**

| Feature                         | Initial Band Steering Implementation  | Wi-Fi SON Implementation  |
|---------------------------------|---|---|
| Utilization measurements        | <ul style="list-style-type: none"> <li>■ Measured every 10 seconds and averaged over 5 minutes (both configurable)</li> <li>■ Overload defined in terms of light and heavy overload thresholds where the light thresholds are used to compute a relative difference between the two bands.</li> </ul> | <ul style="list-style-type: none"> <li>■ Measured every 10 seconds and averaged over 1 minute (both configurable) in the single AP configuration. Measured approximately every 2 seconds and averaged over 30 seconds for a multi-AP configuration.</li> <li>■ Overload defined in terms of just one threshold for each band. The concept of a slight overload has been removed.</li> </ul> |
| STA inactivity detection        | <ul style="list-style-type: none"> <li>■ Based on uplink traffic only (for performance reasons)</li> <li>■ 60 seconds when band is not overloaded; 30 seconds when it is (both configurable)</li> </ul>   | <ul style="list-style-type: none"> <li>■ Based on uplink traffic only (for performance reasons)</li> <li>■ 10 seconds regardless of overload (although the ability to separately configure them remains)</li> </ul>   |
| Airtime fairness integration    | None  | <ul style="list-style-type: none"> <li>■ Avoids steering a client with an ATF reservation on the current BSS</li> <li>■ Prefers steering to a BSS with reserved airtime if the client is currently on a BSS without a reservation</li> </ul>  |
| PHY capability aware offloading | None  | <ul style="list-style-type: none"> <li>■ Optionally (disabled by default) consider the client PHY capabilities when deciding which clients to offload when a channel is overloaded.</li> </ul>  |
| Number of channels supported    | 2   | <ul style="list-style-type: none"> <li>■ Up to 3 (one 2.4 GHz and two 5 GHz)</li> </ul>   |

The most significant change in the Wi-Fi SON implementation is the introduction of 802.11v BSS Transition Management as the preferred method for steering. This standards based mechanism is less disruptive to end clients as it indicates to them the preferred BSS (es) to which to transition. The implication of this is that the AP can make use of it to steer clients that are actively sending data to the AP with only a small interruption in the traffic and limited to no perceptible impact to the application. More mobile devices are beginning to support this standard and the expectation is that this will increase further as STAs seek Multi Band Operation (MBO) certification. Because the way in which STAs respond to this request varies, the implementation incorporates some algorithms to characterize the behavior on a per STA basis and adapt the steering logic accordingly. More details can be found below.

### 16.2.1 Configuration basics

The band steering feature is implemented in a daemon called the Load Balancing Daemon (lbd). In QSDK-based platforms, the `/etc/init.d/lbd` script is responsible for taking the corresponding UCI configuration (found in `/etc/config/lbd`) and generating the format that can be understood by the daemon itself. It also determines the set of Wi-Fi interfaces which should be managed by lbd. By default, all AP interfaces across all radios are provided to lbd for band steering purposes. It can be optionally restricted to a specific SSID through a configuration parameter. More advanced rules to select which interfaces to manage could potentially be implemented in the init script, although none have been done to date.

A few things to note about how lbd manages the configured interfaces in this release. First, it is currently limited to 16 unique SSIDs (although this can be changed at compilation time). Second, in this release, steering is only done within the same ESS (same SSID), as the mechanisms used are most successful only under those conditions. Therefore, to take advantage of band steering, every ESS on which you want band steering to act should have a VAP configured on each band. It is allowed for a given SSID to exist on only one band, although band steering will not make use of it.

For steering between SSIDs, see the *SSID Steering* section.

All configuration parameters are managed using the UCI or the web interface. The UCI configuration is organized into a set of sections which some OEMs may want to tweak and a set of sections which are unlikely to be tweaked by OEMs. These latter sections have an \_Adv suffix and should generally not be changed without consulting with customer engineering. The basic overall configuration parameters are given in [Table 16-2](#)

**Table 16-2 Top-level configuration**

| Configuration Type | Section | Option                 | Description   | Default |
|--------------------|---------|------------------------|---|---------|
| config             | config  | Enable                 | Whether the load balancing logic is enabled or not  | 0       |
| config             | config  | MatchingSSID           | The list of SSIDs to match on which band steering is applicable. To add a list of SSIDs, enter the following command:<br><br>uci add_list<br>lbd.config.MatchingSSID='ssid_to_match'<br>To delete the configured list of SSIDs, enter the following command:<br><br>uci del_list<br>lbd.config.MatchingSSID='ssid_to_match'<br>By default, the MatchingSSID list is empty, which indicates that all the interfaces are to be included. An empty MatchingSSID list and a list that contains matching SSID values cannot coexist. Therefore, to configure interfaces that match a particular SSID, issue the uci del_list lbd.config.MatchingSSID="" command (for example, uci add_list lbd.config.MatchingSSID='ssid_name'). | -       |
| config             | config  | PHYBasedPrioritization | Boolean flag indicating whether the steering logic should attempt to put and/or keep 802.11ac clients on 5 GHz or not.  | 0       |

Refer to the *Wireless LAN Access Point (Driver Version 10.4) Command Reference Guide* (80-Y8052-1) to enable and start the load balancing daemon. After the daemon is enabled, it is restarted whenever the Wi-Fi interface configuration changes.

### General note about RSSI configuration parameters

All RSSI values that can be controlled in the configuration file represent uplink RSSI values in the units reported by the wlanconfig command. Because the term used in other documentation is

RSSI, that same term is used here, although the values appear to be more like an SNR than an RSSI.

## 16.2.2 Types of steering and conditions

There are three primary types of steering supported in this phase. They are pre-association steering, idle post-association steering, and active post-association steering. The pre-association steering is only done during overloaded conditions whereas both flavors of post-association steering can be done based on overload and based on per-STA conditions. Each of these is described in more detail in the sections below.

### 16.2.2.1 Pre-association steering

Pre-association steering aims to connect dual band stations to a channel that is not overloaded. A channel is considered overloaded when its average medium utilization over the span of a minute exceeds 70%. While this condition holds, if the AP sees a dual band STA sending probe requests and the probe requests on the non-overloaded channel exceed a configurable threshold (20 dB), a blacklist is installed and probe responses are withheld on the overloaded channel to encourage the STA to associate on the non-overloaded channel. Note that this logic is slightly different from that in the phase 1 implementation, as now the RSSI is checked even when attempting to pre-association steer to 2.4 GHz. This change was made to account for regulatory domains where the 2.4 GHz maximum transmission power may be less than that on 5 GHz.

Note that clients are only pre-association steered if the conditions which prevent too frequent steering have been met. In addition, the steering safety mechanisms still apply to ensure that STAs being pre-association steered are not orphaned if they are persistently trying to associate to the blacklisted BSS. Refer to Section 7.13.5 for details on these mechanisms.

The basic configuration parameters that are relevant to pre-association steering are shown in [Table 16-3](#).

**Table 16-3 Pre-Association Steering Parameters**

| Configuration Type | Section | Option                  | Description  | Default |
|--------------------|---------|-------------------------|--|---------|
| Offload            | Offload | MUAvgPeriod             | Number of seconds to average before generating a new utilization report on both bands                | 60      |
| Offload            | Offload | MUOverloadThreshold_W2  | Medium utilization threshold (in percentage) for an overload condition on 2.4 GHz                    | 70      |
| Offload            | Offload | MUSOverloadThreshold_W5 | Medium utilization threshold (in percentage) for an overload condition on 5 GHz                      | 70      |
| Offload            | Offload | OffloadingMinRSSI       | Uplink RSSI (in dB) above which pre-association steering and post-association offloading is allowed. | 20      |

### 16.2.2.2 Idle post-association steering

The AP monitors activity for all of its associated STAs by looking for a period of time (by default 10 seconds) during which the STA does not send any non-null uplink data packets. This value was chosen to account for the fact that some STAs will periodically send packets (such as an ARP for the gateway address every 15 seconds) as a form of keep alive. As packet types are not examined (primarily for performance reasons), these keep alive packets can result in a STA never being classified as idle. By using a shorter timer, most of these background/keep alive packets can be ignored.

When a dual band client becomes idle, its uplink RSSI is evaluated to determine if it would be a candidate for steering to a different channel. This evaluation is done by comparing the uplink RSSI on the non-serving channels to a configurable threshold. This RSSI on non-serving channels can be obtained through one of two mechanisms. First, if the client sent enough probe requests on the channel recently, the average value of these probes is used in comparing against the threshold. If not enough probes on the channel have been received, an estimate is computed based on the serving channel RSSI. If the AP has no recent serving channel RSSI information for the client, it will send a series of QoS Null Data packets to estimate its uplink RSSI based on the ACKs for these packets.

The estimate of the non-serving channel RSSI attempts to adjust for typical path loss differences between 2.4 GHz and 5 GHz (in a conservative manner). Unlike in phase 1, this estimation now also considers the delta in the transmission power between the serving and non-serving channels. If the candidate channel has a lower transmission power than the serving channel, this is also used to adjust the estimate. However, if the non-serving channel has a higher transmission power, this is not used to adjust the estimate as many clients will not be able to take advantage of this increased power. This new logic helps account for regulatory domains where there are relatively large differences in maximum transmission power between different bands or sub-bands. The equations used for estimation are shown below, both in their symbolic form and with the default configuration parameters. For the exact equations used, please see the Non-serving Uplink RSSI Estimation section below

Regardless of how the non-serving channel RSSI is obtained, it is then compared to a threshold to determine if steering should take place. A separate threshold is defined for upgrade steering (2.4 GHz to 5 GHz) and downgrade steering (5 GHz to 2.4 GHz) in non-overload conditions. A threshold common across bands is used for idle steering under overload conditions. The default thresholds for non-overload steering effectively disable downgrade steering for two reasons. First, modern Wi-Fi clients generally will roam on their own from 5 GHz to 2.4 GHz once the signal becomes sufficiently weak. Secondly, even at relatively weak RSSIs, the 5 GHz performance is typically better than 2.4 GHz if an 80 MHz channel is used in 5 GHz and only a 20 MHz channel is used in 2.4 GHz. However, these thresholds may be tuned if there is a desire to move a STA to 2.4 GHz earlier even if this may result in a peak throughput drop. The threshold for overload steering allows for both upgrade and downgrade steering.

Once the determination has been made to steer the client, one of two mechanisms can be used. The mechanism used in phase 1 for legacy STAs continues to be supported. This approach first installs a blacklist on the currently serving VAP (and any others besides the target VAP) and then forcefully disassociates the STA. Probe responses are withheld on the blacklisted VAPs until the STA associates again or one of the steering safety mechanisms aborts the steering. If the STA still does try to authenticate with the previously serving VAP, it will be rejected. This is usually enough to encourage the STA to select a different BSS.

The other steering mechanism supported in this release is 802.11v BSS Transition Management (BTM). This is a standard defined mechanism which allows an AP to indicate to a STA that it should move to a new BSS and provide a prioritized list of candidate BSSes. For clients that advertise this capability when associating, the AP will attempt to use this mechanism instead of the legacy mechanism. However, since not all STAs honor the BTM request in the same manner, the AP will also by default use the blacklist and probe response withholding scheme to improve the reliability of the transition. This hybrid approach can be disabled and may become less necessary as STAs become certified by the Multi-Band Operation program in Wi-Fi Alliance.

The basic configuration parameters relevant to idle steering are shown in [Table 16-4](#). Note that these are limited to those that are relevant for non-overload steering. Refer to those in [Table 16-3](#) above for those that are relevant for idle offloading.

**Table 16-4 Basic Idle non-overload steering parameters**

| Configuration Type | Section   | Option               | Description   | Default |
|--------------------|-----------|----------------------|---|---------|
| IdleSteer          | IdleSteer | NormalInactTimeout   | Number of seconds for the inactivity value under no overload conditions on both bands   | 10      |
| IdleSteer          | IdleSteer | OverloadInactTimeout | Number of seconds for the inactivity value when the serving band is overloaded  | 10      |
| IdleSteer          | IdleSteer | InactCheckInterval   | How frequently (in seconds) to check for inactive associated STAs on both bands   | 1       |
| IdleSteer          | IdleSteer | RSSISteeringPoint_DG | The point at which the measured or estimated RSSI on 2.4 GHz dictates a node associated on 5 GHz should be steered to 2.4 GHz.<br>Note that this value effectively disables downgrade steering under non-overload conditions. | 5       |
| IdleSteer          | IdleSteer | RSSISteeringPoint_UG | The point at which the measured or estimated RSSI on 5 GHz dictates a node associated on 2.4 GHz should be steered to 5 GHz.  | 20      |

### 16.2.2.3 Active post-association steering

For clients that support 802.11k and 802.11v, band steering can now take advantage of these standards to steer them while they are actively exchanging data. This was not possible with the legacy steering mechanism due to the time it takes for a STA to associate again. This could often lead to failures in the application. Using 802.11v BSS Transition Management on the other hand, the STAs that support it are able to transition in a much shorter period of time (typically much less than a second) and applications survive the transition with limited impact.

As mentioned in the BTM Compliance Overview section below, in order for a STA to become eligible for active steering, it must first successfully be idle steered using BTM. Once a client is deemed active steering eligible, certain conditions must be met for it to be active steered. For a non-overloaded steer, the trigger conditions are dependent on the serving band. On 2.4 GHz, both an uplink RSSI threshold and a downlink PHY rate threshold must be exceeded. Both conditions are required to ensure that the STA both has a strong enough signal and is not experiencing a high packet error rate (which could also be present on 5 GHz).

On the other hand, for a STA currently being served on 5 GHz, either the uplink RSSI or the downlink PHY rate dropping below the configured threshold is sufficient to start the active steering evaluation process. This more relaxed policy attempts to account for the fact that the PHY rate may stay relatively high even when the RSSI has dropped significantly.

Note that the default configuration sets these downgrade non-overload thresholds so low that downgrade active steering is not triggered. Similar to the idle steering case, this is done because the thresholds would have to be set relatively high for the AP to trigger steering before the STA roaming logic kicks in. However, doing so could lead to reduced peak throughput for that STA, so it is generally better to let the STA make its own decision.

Once a trigger has occurred for non-overload steering, the AP estimates the downlink and uplink throughput for that STA through the use of Tx and Rx byte counters (sampled at the beginning and end of a 1 second interval). At the second sample, the last downlink PHY rate is also obtained and used to compute an estimated airtime on the currently serving channel. The AP then selects a non-overloaded channel (with preference given to those non-overloaded channels on the other band) and requests the STA to perform an 802.11k Beacon Measurement on the candidate channel.

From this downlink RSSI measurement, the AP attempts to estimate an MCS that will be achieved by that client, taking into account both the AP and STA's capabilities on the candidate channel (which includes the bandwidth, number of spatial streams, and phy mode). From this and the previously measured throughput, an airtime value is computed. This value is then used to determine whether the STA can fit on the candidate channel without causing an overload. This is done by adding the estimated airtime to the last measured medium utilization and comparing the result against a safety threshold. If this threshold is not exceeded, the steer is allowed to proceed and the estimated airtime is added to a projected airtime increase that is maintained until a new medium utilization measurement is obtained.

For overload active steering, the trigger is the overload event itself. When this occurs, the first step the AP takes is to estimate the airtime of all active steering eligible clients on the overloaded channel. This is done using the same technique as described above when a single client measurement is triggered. These values are then sorted by airtime in descending order. A candidate channel for offloading is then selected (using the same mechanism described above). Each STA in turn is requested to perform an 802.11k Beacon Measurement to assess its performance on the candidate channel. From this, a decision is made in the same manner as above to either steer it to that channel or not to do so due to the risk of overload.

One additional condition is also applied in that the estimated rate on the target channel must be a configurable amount better than the rate on the current channel when steering to a 5 GHz channel. Once the handling for one STA is completed, consideration then proceeds to the next STA with a new 802.11k Beacon Measurement Request. This process continues until either all active steering eligible clients are exhausted or the amount of airtime estimated to have been removed from the currently overloaded channel would cause the medium utilization to fall below the safety threshold.

Any time active steering is performed (either for offloading purposes or due to an individual STA's RSSI and/or MCS values crossing the thresholds), the immediately following medium utilization measurement triggers a steering blackout period. During this period, no active upgrade steers are allowed so that a more accurate assessment of the impact (to medium utilization) of previous active steers can be obtained without further active steers adding uncertainty to the data.

Active downgrade steers are still permitted (although they will not be triggered with the default configuration) as their primary purpose is to ensure STAs can maintain connectivity rather than to improve performance. Idle steers are also permitted during this blackout as these clients are not currently active and thus should not impact the utilization measurements unless they become active.

The basic parameters that control active steering are provided in [Table 16-5](#).

**Table 16-5 Basic active steering configuration parameters**

| Configuration Type | Section     | Option                   | Description  | Default |
|--------------------|-------------|--------------------------|--|---------|
| ActiveSteer        | ActiveSteer | TxRateXingThreshold_UG   | The rate (in Kbps) at which a rate crossing event should be generated for a potential active client upgrade to 5 GHz.<br>Note that due to implementation reasons (integer division), the rate used for the triggers will be an underestimate of the actual rate.     | 50000   |
| ActiveSteer        | ActiveSteer | RateRSSIXingThreshold_UG | The value (in dB) the uplink RSSI must be above to be considered for active steering to 5 GHz.<br>This threshold is in conjunction with the TxRateXingThreshold_UG.  | 30      |
| ActiveSteer        | ActiveSteer | TxRateXingThreshold_DG   | The rate (in Kbps) at which a rate crossing event should be generated for a potential active client downgrade to 2.4 GHz.<br>Note that due to implementation reasons (integer division), the rate used for the triggers will be an underestimate of the actual rate. | 6000    |
| ActiveSteer        | ActiveSteer | RateRSSIXingThreshold_DG | The value (in dB) the uplink RSSI may be below to be considered for active steering to 2.4 GHz.<br>This threshold is an additional trigger (an OR condition) for downgrade (with the other trigger being the TxRateXingThreshold_DG).                                | 0       |
| Offload            | Offload     | MUSafetyThreshold_W2     | The percentage of medium utilization that the measured plus projected utilization is allowed to reach before all further downgrade or offloading steering is disallowed until a new utilization measurement is done.   | 50      |
| Offload            | Offload     | MUSafetyThreshold_W5     | The percentage of medium utilization that the measured plus projected utilization is allowed to reach before all further upgrade or offloading steering is disallowed until a new utilization measurement is done.   | 60      |

#### 16.2.2.4 Interference Avoidance Steering (IAS)

In a Wi-Fi environment, there may be scenarios where an associated STA suffers from downlink throughput degradation due to a node that is hidden from the AP. This occurs because the AP thinks the medium is idle (CCA checks pass) even though that is not the case where the STA is located. Such interference can often be inferred at the AP by observing certain runtime performance parameters that are available at the AP. Specifically, it is possible to examine the relationship between downlink rate (MCS) and uplink RSSI to detect cases where the STA is being impacted by interference.

Interference Avoidance Steering (IAS) adds new rules for triggering steering of a STA to a different channel and/or band under such interference. When a STA is associated to an AP, the Load Balancing Daemon (lbd) examines each new uplink RSSI and downlink rate (MCS) value it receives from the driver. Whenever there is sufficient downlink traffic, the RSSI and MCS are fed to an interference detection algorithm, usually referred to as the ‘detector’.

The detector has been trained (through extensive data collection and machine learning techniques) to declare whether a single sample (Uplink RSSI, Downlink rate) reflects likely interference or not. A separate instantiation of this algorithm is available for the following STA capabilities:

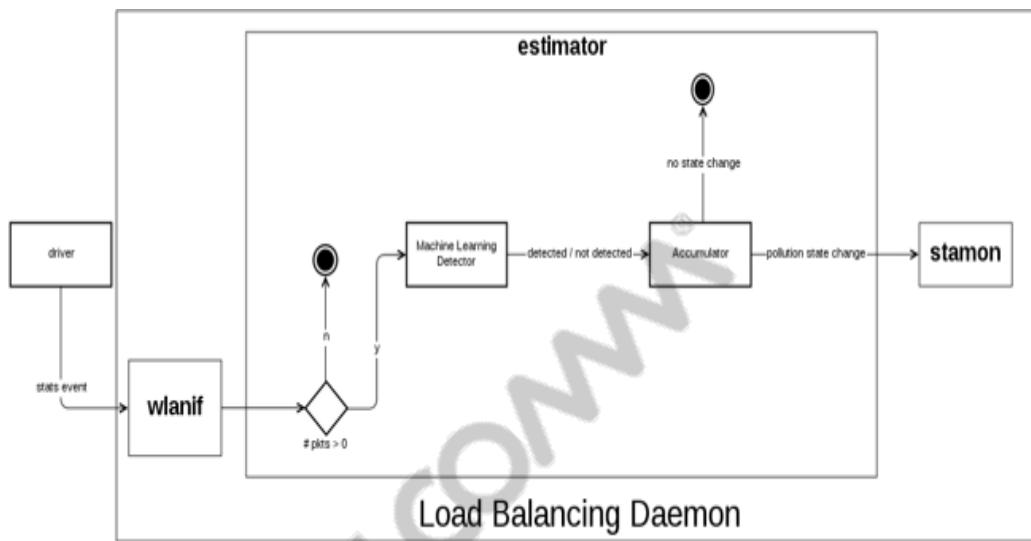
- 2.4 GHz one spatial stream at 20 MHz
- 2.4 GHz two spatial streams at 20 MHz
- 5 GHz one spatial stream at 40 MHz
- 5 GHz two spatial streams at 40 MHz
- 5 GHz one spatial stream at 80 MHz
- 5 GHz two spatial streams at 80 MHz

For cases where a STA is more capable than one of the capabilities available above, the one that is considered the closest match is used, where the matching is done first by band, then by bandwidth and number of spatial streams.

Each output from the interference detector is fed to an additional stage in the processing pipeline referred to as the accumulator. The accumulator does smoothing of the individual samples and only declares that there is interference or there is no longer interference if a configurable number of samples is met. With the default configuration, if 60% or more of the last 5 samples indicate interference, the accumulator will output ‘interference’.

Once interference is detected, if the number 40% or fewer of the last 5 samples indicate interference, the accumulator will output ‘no interference’. To ensure that samples from the distant past are not grouped together with recent samples, the accumulator requires that all samples being considered must be within a configurable number of seconds multiplied by the size of the accumulator’s circular buffer (15 seconds \* 5 samples = 75 seconds by default).

The full interference detection pipeline can be seen in the following figure.



**Figure 16-2 Interference Detection Pipeline**

Whenever the accumulator outputs a value, this is considered a potential pollution state change. A BSS is considered polluted whenever there has been a sufficient number of interference detected samples output from the machine learning detector within a period of time. After a BSS is declared as polluted, the *stamon* module within the Load Balancing Daemon is informed so that it can potentially steer the client to a new channel or band.

If the STA is considered active steering eligible, this will trigger an 802.11k measurement and the selection of the new BSS will proceed as for other steering triggers. If the STA is not considered active steering eligible but is idle steering eligible and is currently idle, the selection of a new BSS will follow the same rules as those used when the current channel is considered overloaded. In other words, if the serving RSSI is above the MinOffloadingRSSI value, any non-overloaded non-polluted BSS is considered.

Note that in all forms of steering, any BSS that is marked as polluted is ignored. Furthermore, when selecting a channel on which to perform an 802.11k Beacon Measurement, channels with no polluted BSSes will be preferred over those that have at least one polluted BSS.

A polluted BSS can be cleared through one of two means. First, if the STA is actually associated to the polluted BSS, if the accumulator outputs “no interference” then the pollution will be cleared. Second, if *MaxPollutionTime* seconds elapse without any further “interference” output from the accumulator (for example, if insufficient samples are present or if the STA not being associated on that BSS), then the pollution state is automatically cleared.

The basic parameters that control Interference Avoidance Steering are provided in [Table 16-6](#).

**Table 16-6 Basic Interference Avoidance Steering configuration parameters**

| Configuration Type | Section | Option           | Description   | Default           |
|--------------------|---------|------------------|---|-------------------|
| IAS                | IAS     | Enable_W2        | Whether to enable Interference Avoidance Steering on 2.4 GHz.<br>If this is set to 0, no attempt will be made to detect interference for a STA associated on 2.4 GHz.                                   | 1                 |
| IAS                | IAS     | Enable_W5        | Whether to enable Interference Avoidance Steering on 5 GHz.<br>If this is set to 0, no attempt will be made to detect interference for a STA associated on 5 GHz.                                       | 1                 |
| IAS                | IAS     | MaxPollutionTime | The number of seconds after which a BSS that was previously marked as polluted is considered no longer polluted.<br>Note that a BSS may also have its pollution cleared by output from the accumulator. | 1200 (20 minutes) |

### 16.2.2.5 Interference as an input to band steering (Wi-Fi SON) for AR938x

Netlink messages to load balancing daemon (LBD) must be sent with certain information about the stations connected to the AP. This transmission of netlink messages enables the LBD logic running in the AP to take a decision regarding the presence or absence of interference and accordingly steer the client to the other band or let it be. The information that is periodically sent to LBD include the client's RSSI, txrate, tx packet count, rx packet count, tx byte count, rx byte count and rx packet count. This periodic transmission of information to LBD was performed in the offload path. Starting with QCA\_Networking\_2016.SPF.4.0, this functionality is also introduced in the direct attach path.

With the required statistics available, the LBD logic will function effectively even in the direct attach path to detect interference. The source of interference is hidden to the AP but affects the station(s) connected to it. The interference detection behavior exhibited by the LBD logic is consistent regardless of the hardware being direct attach or offload.

A timer function is designed in the direct attach path, that will send periodic updates of client stats to LBD as netlink messages. This periodic nature has been kept to mimic the periodic nature of WMI events in the offload path. The default value of this timer is 500 msecs but can be user configured.

The timer is initialized the same time when band steering is initialized for direct attach and is freed at the time band steering is disabled. The handler function of the timer (wlan\_bsteering\_interference\_stats\_event\_handler) does the following:

1. It iterates through the list of all clients connected and saves their stats-one client at a time-to the bs\_sta\_stats\_ind structure.
2. Once the number of clients the bs\_sta\_stats\_ind structure has been updated for reaches 3, or the present client being processed is connected to a different VAP, the netlink message is sent to LBD with the bs\_sta\_stats\_ind structure constituting the data part of the message.

3. If interference detection is not enabled, the event is not sent to LBD and the `bs_sta_stats_ind` structure is not updated.
4. The timer re-arms itself and continues once all the clients have been processed.

### 16.2.3 BTM compliance overview

Given that there is significant variation in how well various STA implementations respond to 802.11v BSS Transition Management (BTM) requests, new logic has been incorporated into band steering to classify the behavior. At a very high level, the logic consists of the following:

- Require that BTM based steering under idle conditions succeed before even attempting active steering.
  - This is done under the assumption that a STA that rejects or otherwise fails to move to the desired BSS under idle conditions is more likely to do the same when active.
- If idle BTM steering fails, revert to legacy steering and consider the device as BTM unfriendly for a time (with exponential backoff if idle BTM steering keeps failing).
- If BTM active steering repeatedly fails, do not perform active steering again until both an active steering unfriendliness timer expires (which has exponential backoff in case it keeps failing) and then a BTM idle steer succeeds.
- If the uplink RSSI falls below a threshold (by default 12 dB) on the serving channel, use BTM based steering without blacklists and do not count a failure against the client.
  - This is a conservative approach in case the estimate of the RSSI on the non-serving channel was inaccurate.
- If the client accepts the BTM request but specifies a different BSSID, use BTM based steering without blacklists and do not count a failure against the client.
  - This helps account for environments with multiple APs operating within the same ESS where a client may see a stronger AP and thus decide to transition to it.

The parameters relevant to the BTM compliance logic are provided in [Table 16-7](#).

**Table 16-7 BTM compliance configuration parameters**

| Configuration Type | Section        | Option             | Description   | Default |
|--------------------|----------------|--------------------|---|---------|
| SteerExec          | SteerExe c_Adv | BTMResponseTime    | Maximum response delay for 802.11v BSS Transition Management Request.   | 10      |
| SteerExec          | SteerExe c_Adv | BTMAssociationTime | The maximum time allowed for an 11v-capable client to reconnect before AP aborts steering the client and releases the blacklist for the client. | 6       |
| SteerExec          | SteerExe c_Adv | BTMAAlsoBlacklist  | Whether non-best effort steering should use blacklists in addition to the actual BTM messaging.   | 1       |

**Table 16-7 BTM compliance configuration parameters**

| Configuration Type | Section        | Option                  | Description  | Default            |
|--------------------|----------------|-------------------------|--|--------------------|
| SteerExec          | SteerExe c_Adv | BTMUnfriendlyTime       | The time period to wait prior to steering a 11v-capable client again upon BTM steering failures (subject to exponential backoff).  | 600                |
|                    |                |                         | <b>WARNING</b> Care should be taken when reducing this parameter as it could reduce the probability of successful steering for a device that typically rejects or does not respond to BTM. |                    |
| SteerExec          | SteerExe c_Adv | MaxBTMUnfriendly        | Maximum time (in seconds) for the BTM steering unfriendly timer for idle steering.   | 86400<br>(1 day)   |
| SteerExec          | SteerExe c_Adv | MaxBTMActiveUnfrien dly | Maximum time (in seconds) for the BTM steering unfriendly timer for active steering.   | 604800<br>(1 week) |
| SteerExec          | SteerExe c_Adv | MinRSSIBestEffort       | The RSSI below which BTM-based steering will operate in best effort mode (where no blacklists are installed).  | 12                 |

### 16.2.4 Steering safety and hysteresis

The band steering implementation implements some safety and hysteresis mechanisms to ensure STAs do not decide to switch to cellular from Wi-Fi or otherwise blacklist the AP due to steering. These, coupled with the conservative estimation thresholds, also help provide a basic degree of hysteresis to avoid STAs from being steered too frequently. At a high level, the following mechanisms exist.

1. Separate timers for legacy and BTM-based steering for how frequently a STA can be steered. When a STA is steered, this prohibit timer is started and prevents further steering attempts until it expires.
2. When using the legacy steering approach or BTM steering with blacklists (as is the default), abort the steering if the STA tries to authenticate on the old channel too many times.
3. Maximum amount of time to allow for a STA to associate after being steered before declaring a failure. There are separate timers for legacy and BTM steering.
4. RSSI thresholds below which the steering is aborted without declaring it a failure. These help handle scenarios where the RSSI estimate was overly optimistic or the STA's signal degrades quickly (eg. due to mobility).
5. Exponential backoff for repeated steering failures.

The configuration parameters relevant for these mechanisms are shown in [Table 16-8](#).

**Table 16-8 Configuration parameters**

| Configuration Type | Section       | Option                       | Description   | Default            |
|--------------------|---------------|------------------------------|---|--------------------|
| SteerExec          | SteerExec     | SteeringProhibitTime         | <p>Number of seconds to wait prior to steering the client again after a successful legacy steer or an attempt by a STA to authenticate on a blacklisted VAP.</p> <p><b>WARNING</b> This value should not be reduced other than for demo purposes as too short of a value can lead to the STA blacklisting the AP.</p> | 300                |
| SteerExec          | SteerExec     | BTMSteeringProhibitShortTime | The time period to wait prior to steering an 11v-capable client again after a successful steering within BTMAssociationTime.  | 30                 |
| SteerExec          | SteerExec_Adv | TSteering                    | Number of seconds allowed for the client to reconnect before AP aborts steering   | 15                 |
| SteerExec          | SteerExec_Adv | SteeringUnfriendlyTime       | <p>The amount of time a device is considered steering unfriendly before another attempt.</p> <p><b>WARNING</b> Care should be taken before reducing this value as too small of a value may increase the likelihood of the STA black listing the AP.</p>   | 600                |
| SteerExec          | SteerExec_Adv | MaxSteeringUnfriendlyTime    | Maximum time (in seconds) for the legacy steering unfriendly timer.   | 604800<br>(1 week) |
| SteerExec          | SteerExec_Adv | TargetLowRSSIThreshold_W2    | RSSI threshold (in dB) indicating 2.4 GHz band is not strong enough for association   | 5                  |
| SteerExec          | SteerExec_Adv | TargetLowRSSIThreshold_W5    | RSSI threshold (in dB) indicating 5 GHz band is not strong enough for association   | 15                 |
| SteerExec          | SteerExec_Adv | LowRSSIXingThreshold         | RSSI threshold to generate an indication when a client crosses it (in dB)   | 10                 |
| SteerExec          | SteerExec_Adv | Delay24GProbeRSSIThreshold   | The minimum RSSI threshold to delay probe responses in 2.4 GHz band (in dB)   | 35                 |
| SteerExec          | SteerExec_Adv | Delay24GProbeTimeWindow      | The time window within which probe responses will not be sent for configured count(s)   | 0                  |
| SteerExec          | SteerExec_Adv | Delay24GProbeMinReqCount     | The probe request count above which probe responses will be sent for all received within time window  | 0                  |

## 16.2.5 Advanced configuration parameters

The parameters shown in [Table 16-9](#) may also be changed, although some of them have been carefully selected and OEMs are encouraged to leave them unchanged.

**Table 16-9 Advanced configuration parameters**

| Configuration Type | Section         | Option                | Description  | Default           |
|--------------------|-----------------|-----------------------|--|-------------------|
| config             | config_Adv      | AgeLimit              | The maximum age (in seconds) for measured values before they are considered too out of date from which to make a steering decision.  | 5                 |
| StaDB              | StaDB           | IncludeOutOfNetwork   | Whether out of network devices should be included in the database or not.  | 1                 |
| StaDB              | StaDB_Adv       | AgingSizeThreshol d   | The number of entries allowed in the station database before periodic aging is triggered.  | 100               |
| StaDB              | StaDB_Adv       | AgingFrequency        | Once aging is triggered, how frequently (in seconds) to perform aging of the station database.   | 60                |
| StaDB              | StaDB_Adv       | OutOfNetworkMax Age   | The number of seconds that must elapse since the last update for an out-of-network entry before it is considered too old and is removed from the database.   | 300               |
| StaDB              | StaDB_Adv       | InNetworkMaxAge       | The number of seconds that must elapse since the last update for an in-network entry before it is considered too old and is removed from the database. Only unassociated entries will be considered for removal. | 2592000 (30 days) |
| StaMonitor         | StaMonitor_Adv  | RSSIMeasureSamples_W2 | Number of RSSI measurements to average using NDP before generating a RSSI report on 2.4 GHz  | 5                 |
| StaMonitor         | StaMonitor_Adv  | RSSIMeasureSamples_W5 | Number of RSSI measurements to average using NDP before generating a RSSI report on 5 GHz  | 5                 |
| BandMonitor        | BandMonitor_Adv | MUCheckInterval_W2    | How frequently (in seconds) to check the medium utilization on 2.4 GHz   | 10                |
| BandMonitor        | BandMonitor_Adv | MUCheckInterval_W5    | How frequently (in seconds) to check the medium utilization on 5 GHz   | 10                |
| BandMonitor        | BandMonitor_Adv | ProbeCountThreshold   | The number of consecutive probe request RSSI values that must be available to consider using the average RSSI when making pre-association steering decisions.  | 1                 |
| Estimator_Adv      | Estimator_Adv   | RSSIDiff_EstW5FromW2  | Difference when estimating 5 GHz RSSI value from the one measured on 2.4 GHz.  | -15               |
| Estimator_Adv      | Estimator_Adv   | RSSIDiff_EstW2FromW5  | Difference when estimating 2.4 GHz RSSI value from the one measured on 5 GHz.  | 5                 |
| Estimator_Adv      | Estimator_Adv   | ProbeCountThreshold   | The number of consecutive probe request RSSI values that must be available to consider using the average RSSI on the unassociated band when making steering decisions.   | 3                 |

**Table 16-9 Advanced configuration parameters (cont.)**

| Configuration Type | Section       | Option                     | Description  | Default |
|--------------------|---------------|----------------------------|--|---------|
| Estimator_Adv      | Estimator_Adv | StatsSampleInterval        | The amount of time (in seconds) between consecutive samples of the byte count statistics for a STA when estimating its data rate.  | 1       |
| Estimator_Adv      | Estimator_Adv | 11kProhibitTime            | The minimum amount of time (in seconds) to enforce between consecutive 802.11k Beacon Requests.  | 30      |
| Estimator_Adv      | Estimator_Adv | PhyRateScalingForAirtime   | The factor by which to scale the estimate PHY rate to arrive at an approximate effective MAC rate.   | 50%     |
| Estimator_Adv      | Estimator_Adv | EnableContinuousThroughput | Run with throughput sampling always enabled (for demo purposes only).  | 0       |
| Estimator_Adv      | Estimator_Adv | BcnrptActiveDuration       | Duration (in milliseconds) for an active mode 802.11k Beacon Request   | 50      |
| Estimator_Adv      | Estimator_Adv | BcnrptPassiveDuration      | Duration (in milliseconds) for a passive mode 802.11k Beacon Request   | 200     |
| SteerAlg_Adv       | SteerAlg_Adv  | MinTxRateIncreaseThreshold | Minimum amount the 5 GHz PHY rate must be above the 2.4 GHz PHY rate when determining if the channel is good enough.<br>This is only used in overload scenarios.   | 53      |
| SteerAlg_Adv       | SteerAlg_Adv  | MaxSteeringTargetCount     | The maximum number of candidate BSSes for any given steering operation.<br>Currently, this parameter is limited to a value of one because many STAs do not appear to make effective use of the preference value included in 802.11v RSS Transition Management Request. | 1       |

## 16.2.6 Logging and debug CLI

### 16.2.6.1 Debug CLI

To assist test teams and OEMs in evaluating the behavior of the band steering implementation, a debug CLI is provided in the default build of the Load Balancing Daemon (lbd). This CLI has commands which can be used to query the state of the daemon and to trigger steering manually.

It can also be used to obtain the debug logs. To access the debug CLI, connect to the AP (using telnet, SSH, or serial) and then telnet to port 7787 using the loop back IP address (127.0.0.1). To disconnect, use the q command or the normal mechanism of your telnet client to disconnect.

**NOTE** The exit command must not be used because this command terminates the daemon itself.

**NOTE** Open a Telnet session to the APUT on port no.7787 for the Band Steering daemon (lbd) debug logs. Open a Telnet session to the APUT on port no.7777 for the -Hy-Fi daemon (hyd) debug logs.

Commands are grouped by the modules that implement the handling of them. Some of the most relevant ones are described in the sub-sections below. For the complete set of commands, use the help mechanism in the debug CLI or contact customer engineering.

## Caveats

This debug CLI is primarily intended for testing purposes. Consideration should be given before shipping a product as to whether it should remain enabled in an end product. It can be compiled out by building lbd with the LBD\_DBG\_MENU make variable set to n.

### dbg module

The dbg module is the one responsible for enabling/disabling the debug logging to the debug CLI connection. To enable logging, use the dbg here command. Note that once logging is enabled, you may want to create a new connection for any further interactions as the logs will be intermingled with the CLI prompts.

Logs are grouped into functional areas, with each functional area having a separate logging level. The levels, in decreasing verbosity, are: dump, debug, info, and ERR. To view the current levels, use the dbg level command. To change the level, use the dbg level <area name> <level name> command.

[Table 16-10](#) shows the available log areas.

**Table 16-10 Available debug logging areas**

| Area      | Description  | Default Log Level |
|-----------|--|-------------------|
| wlanif    | Logs for events coming from the Wi-Fi driver (before they have been handled by other modules).   | debug             |
| steerexec | Logs for the steering executor. These logs indicate when a station completes steering or is marked as eligible for steering again.   | info              |
| steeralg  | Logs for the steering algorithm (responsible for steering decision making). These logs indicate when a STA is evaluated to determine if it's eligible for steering, and how BSSes are selected as steering candidates. | info              |
| stamon    | Logs for the station monitor log area. These logs indicate when a device is being steered during post-association.   | info              |
| stadb     | Logs for the station database. There is not much interesting in this area, other than indications when the debug CLI is used to change state.  | info              |
| ratestats | Logs for the continuous throughput statistics (for demo GUI purposes).   | ERR               |
| probe     | Logs for probe request events. These are generated at dump level and happen frequently and should only be enabled if needed for verification purposes.   | ERR               |
| lbd       | Logs for the main event loop of the daemon. There is not much interesting here other than a log that indicates it is running.  | debug             |
| estimator | Logs for the rate estimator. These logs provide info on the STA data rate, MCS, and airtime.   | info              |
| bandmon   | Logs for the band monitor log area. These logs indicate when a band is considered overloaded and when a station is being steered pre-association.  | info              |

To disable logging to the current CLI connection, use the following command

```
dbg here -off
```

## stadb module

The *stadb* module provides information about the currently and previously associated STAs. The primary command in this module is the *stadb s* command. This will display the state of all in-network STAs (where an in-network STA is one that has/was associated since the load balancing daemon was started). An example output is shown below:

```
@ stadb s
Num entries = 148
MAC Address          Age      Bands    Assoc? (age)
Active? (age)        Flags
00:EE:BD:9A:68:BA   4074     25       APId 255 ChanId 161 ESSID 0
(6406)   yes (4411)           BTM RRM
D8:D3:85:44:D9:36   6406     2        APId 255 ChanId 1   ESSID 0
(598)   no (28)              RRM
78:4B:87:66:A3:E0   4        25       APId 255 ChanId 149 ESSID 0
(119)   yes (119)            BTM RRM RA
```

The Flags column can contain a few different strings. Clients that support 802.11v BSS Transition Management are denoted with ‘BTM’, and that support 802.11k are denoted with ‘RRM’. A STA that has reserved airtime on the serving BSS is indicated with ‘RA’. A STA that is current in static Spatial Multiplexing Power Save (SMPS) mode is indicated with ‘PS’. Lastly, an MU-MIMO capable STA is denoted with ‘MU’. These ‘PS’ and ‘MU’ designations are used to select the right interference detection curve to reduce the chance of false positives due to 1 spatial stream transmissions being used.

To view information about specific BSSes on which the RSSI and/or reserved airtime is recorded for a given STA as well as which BSSes are marked as polluted, use the *stadb s bss* command. The *stadb s phy* command will show the PHY capabilities reported by this STA at association time. The *stadb s rate* command will show the last rate information estimated/computed by this AP. All three of these commands can be limited to a single STA by specifying its MAC address at the end of the command. Additionally, the complete information for a STA can be obtained through the *stadb s <mac>* command.

## bandmon module

The *bandmon* module provides information about the last measured medium utilization and whether one or both of the channels is considered overloaded. It also displays the projected utilization increase for any active steering attempts that are ongoing or complete but for which a new utilization measurement is not yet available. To display this information, use the *bandmon s* command.

Example output is shown below:

```
@ bandmon s
Channel 1 : Measured: 90 %* (overloaded)  Projected Increase: 0 %
Channel 149: Measured: 5 %*                 Projected Increase: 0 %
```

One thing to note in this output is the asterisk next to the % for the measured utilization value. As overload decisions are made only when a measurement is available on all channels, either the presence or absence of an asterisk for both channels is the indication that all values were measured around the same time. If not all channels are showing or not showing an asterisk, then a measurement is pending from one or more of them.

## steerexec module

The *steerexec* module provides a view into the actual steering execution mechanisms. The *steerexec s* command can be used to get the current status of steering for all STAs that *steerexec* has attempted to steer. An example is shown below.

```
@ steerexec s
Legacy overall state:
    Current # STAs prohibited from steering: 1
    Next prohibit update: 7 seconds
    Current # STAs marked as steering unfriendly: 1
    Next unfriendly update: 301 seconds
    Current # STAs blacklisted: 1
    Next blacklist update: 144 seconds

    Legacy per STA information:
    MAC          Transaction Secs since steered # Auth Rej
Prohibited     Unfriendly      T_Steer      Blacklist type Consecutive
Failures
    22:1B:3A:D2:EC:F3 0          4           0           Short
no            10             Channel        0
    24:46:17:09:DF:A1 1          0           1           Long
yes           Candidate       1

    BTM overall state:
    Current # STAs marked as BTM unfriendly: 1
    Next BTM unfriendly update: 290 seconds
    Current # STAs marked as BTM active unfriendly: 1
    Next BTM active unfriendly update: 9601 seconds

    802.11v BTM Compliant per STA information:
    MAC          Transaction Secs since steer(active failure) State
Unfriendly Compliance   Eligibility Token Timer
    22:1B:3A:D2:EC:F3 0          4           (0)           WaitAssoc
no            Idle           Idle           6
    24:46:17:09:DF:A1 1          0           (0)           Idle           yes
ActiveUnfriendly None      2

    802.11v BTM Compliant per STA statistics:
    MAC          NoResp Reject NoAssoc Success Consecutive Failures
(active)
    22:1B:3A:D2:EC:F3 0          34          0          33          0           (32 )
    24:46:17:09:DF:A1 0          2           0           1           1           (1 )
```

This provides statistics on the steering attempts for each of the various clients. In addition, it provides indications of the current state of each client with respect to steering and how much time must elapse before certain conditions are re-evaluated. There is too much detail to explain here, but a customer engineer can provide further info as requested.

The *steerexec steer <mac> <ap id> <channel> 0* command can be used to forcefully steer a STA to a specific channel on a specific AP. When using single AP band steering, the *<ap id>* parameter should always be specified as 255 (indicating self). The steering will use the appropriate mechanism (BTM or legacy) based on the client's capabilities and current compliance state. One thing to note is that it does not prevent steering an active STA if it is not active steering eligible, so it should be used only when this is acceptable. It does however check that the STA has not been steered too recently and will reject the request if it has.

## estimator module

The estimator module is responsible for performing airtime estimates using 802.11k Beacon Measurement Requests. To see the current state of all STAs for which a measurement has been requested at some point in the past, use the estimator s command. Example output is shown below.

```
@ estimator s
MAC Address      Throughput State      802.11k State
802.11k Expiry (s)
8C:FD:F0:01:41:22  Idle                802.11k prohibited
27.486348
```

This shows whether the STA is currently prohibited from having an 802.11k measurement done (and thus cannot be active steered) and the duration that is remaining before a new 802.11k request can be sent. This throttles the requests as some STAs do not respond favorably to frequent 802.11k requests and also serves as another hysteresis mechanism.

## 16.2.7 Band steering architecture

The band steering implementation consists of two components, a driver module and a user space daemon (lbd). The driver module uses net link events to inform the daemon of significant events. The intelligence of when to steer and how to react to the behavior of STAs when they are steered resides in the daemon. See the sub-sections below for further details on each of these components.

### 16.2.7.1 Band Steering Driver

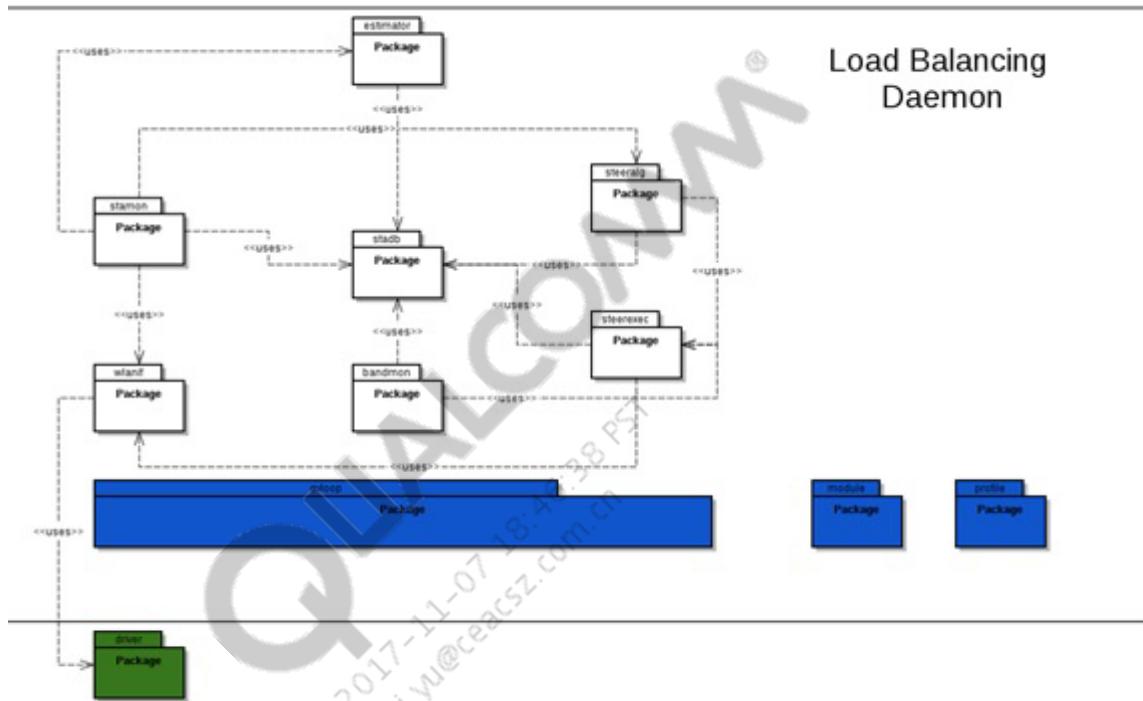
The driver portion of band steering is primarily implemented in a new umac module named band\_steering. Its primary responsibilities include:

- Periodically using the acsreport mechanism to measure the channel utilization, averaging this over a configurable period;
- Reporting STA associations along with the capabilities (PHY and 802.11k/v) to user space;
- Monitoring the RSSI and MCS information for STAs and informing user space when specific thresholds are crossed;
- Monitoring the activity of STAs and informing user space when they become idle or active;
- Reporting to user space when probe requests are received or authentication attempts are rejected due to a blacklist; and
- Forwarding 802.11k and 802.11v responses up to user space.

The band steering logic is disabled by default. When the user space daemon is started, it provides a series of configuration parameters to the driver that control how each of these mechanisms is implemented. The daemon then enables band steering on each of the VAPs it is managing. Only those VAPs where band steering is enabled will result in events being generated for any of the above conditions.

### 16.2.7.2 Load Balancing Daemon

The Load Balancing Daemon (lbd) implements the band steering algorithms themselves. This daemon uses a single threaded event driven framework, reacting to timers, events from the kernel (in the form of netlink messages), and events between modules within the daemon itself. The various modules that make up the daemon are shown in [Figure 16-3](#).



**Figure 16-3 Load balancing daemon high-level architecture**

The responsibility of each of the core modules is described in the following subsections. To support both single AP and multi-AP use cases, each of these modules is broken into three parts: a common part that is shared across both use cases, a part specific to single AP use cases (typically with a BSA suffix in the filename), and a part that is used in multi-AP use cases (with an MBSA suffix). Only the first two parts are located within the **lbd** source tree. The multi-AP specific code is located within the **hyd** source tree.

The packages in blue provide common services that these implementation modules utilize. So as to not clutter the diagram, the usage dependencies on these core packages are not shown.

evloop provides the main event loop which includes facilities for timers and socket event handling. module provides an event dispatch mechanism for the different modules within lbd to deliver asynchronous events to one another. profile provides the means to retrieve parameters from a configuration file (defined in an INI format with named sections corresponding to each package), with default values provided if no entry is found in the configuration file.

## Configuration Parameter Mapping from UCI to lbd

The exact names of the configuration parameters in the file read by lbd differ from those in the UCI configuration file (/etc/config/wireless). The UCI parameters are named more according to their purposes whereas those in the lbd configuration file are named more according to their use in the implementation. Refer to the /etc/init.d/lbd init script to understand how the lbd config parameters are mapped/derived from the UCI ones.

### wlanif module

The *wlanif* module (short for WLAN Interface) is responsible for all interactions with the Wi-Fi driver. This allows the details of the driver interface to be abstracted away such that the other modules can focus on the algorithm details instead of the intricacies of the driver interface. *wlanif* has a few primary roles in the overall lbd implementation. The first is to initialize the driver with the desired configuration and manage the enabling/disabling of band steering. At startup, *wlanif* ensures that band steering is only enabled when the Channel Availability Check (CAC) has completed on a DFS channel and Automatic Channel Selection is not active. Whenever a managed VAP goes down or changes channel, *wlanif* disables band steering in the driver and only re-enables it when CAC and ACS are complete. This helps ensure that the utilization measurements are aligned across the various radios.

The second responsibility is to receive events from the driver and forward them on to other modules. It does this by listening for two netlink event types. The first is the standard RTMGRP\_LINK event, which carries struct iw\_events for operations such as a disassociation of a STA or a channel change. The second is a new NETLINK\_BAND\_STEERING\_EVENT type for all of the band steering specific events. This includes things like association (with capability information), RSSI and MCS crossing, utilization, and a number of others as well. These events are converted to types internal to lbd and injected into the event dispatching mechanism for other modules to handle.

The third responsibility of *wlanif* is to provide APIs for the synchronous commands to the driver. This includes commands to send an 802.11k Beacon Measurement Request, send an 802.11v BSS Transition Management Request, and enable/disable/sample statistics for a given STA. These APIs internally perform the appropriate ioctl and pass the status back to the caller.

### SSID limitations

As noted above, the internal state of *wlanif* is statically sized for up to 16 unique SSIDs and up to 3 radios. Any attempt to configure more than this will result in an error upon startup.

### stadb module

The *stadb* module (short for Station Database) is the central place where all information about specific STAs is stored. It consists of a hash table indexed by MAC address. The hash table has a size of 256 slots by default and uses chaining when there is a collision.

Instead of each module having to define its own state storage mechanism, each entry provides opaque fields where certain modules can store the information that needs to be maintained across upcalls from the event loop. Currently this consists of the state for the steering executor (steerexec), the estimator, and another module that is used only for multi-AP coordinated steering. Other

modules have no need for storing per-STA state information beyond that already made available via the `stabd` public API.

Each in-network entry in the database contains various metrics and capabilities, including the following:

- Which bands the STA is known to operate on
- Where the STA is currently associated or was last associated
- Whether the STA is currently considered active or not
- Overall capability information
  - 802.11k/v support
  - Whether the STA has reserved airtime
  - The best PHY mode across all bands supported by the STA
  - Measured downlink and uplink data rate information Per BSS, the following is maintained
    - PHY capabilities (number of spatial streams, maximum bandwidth, and PHY mode)
    - Last uplink RSSI value (which could be an estimate or based on probe requests)
    - The estimated MCS on the downlink along with the last estimated airtime.
    - The amount of airtime reserved (if any)

Some of the information stored in the station database is updated directly based on the events it receives from `wlanif`. It also offers APIs to allow other modules (such as estimator) to update state.

Additionally, to avoid having to rely on the event dispatch mechanism to deliver events in a specific order, it also offers APIs through which other modules can register for a callback when a specific event occurs, including association changes, RSSI updates, and activity updates. Lastly, it provides a means for other modules to iterate over the station database.

By default, entries are created in the station database when ever an event occurs and they are not already present. This includes events such as probe requests which may be received from STAs that never intend to associate with this AP. Such STAs are considered as out-of-network STAs unless/until they associate. By default, these out-of-network STAs are stored to help with the dual band detection logic (in case they do associate, thereby becoming in-network STAs).

**NOTE** The entries for such out-of-network STAs are smaller in size to reduce memory consumption.

If the database size grows too large, out-of-network STAs are aged out quicker than in-network ones. It is also possible to disable the storage of out-of-network STAs (at the cost of it taking longer to detect whether a STA is dual-band capable).

### Non-persistence

Although this module is named the station database, no information is stored persistently on the file system. Therefore, whenever `lbd` is restarted, all information about STAs is relearned afresh.

## bandmon module

The *bandmon* module (short for band monitor) tracks the utilization and overload status for each of the active channels. Furthermore, when a channel is overloaded, it implements the pre-association steering logic. This consists of two parts. First, when the channel becomes overloaded, it iterates over the station database to determine if any in-network but not currently associated STAs have a recent enough RSSI to be pre-association steered. For each one that does that meets the threshold for pre-association steering, it will call into *steerexec* to perform the blacklisting for pre-association steering. Secondly, for any subsequent RSSI updates for an in-network but unassociated STA while a channel is overloaded, it evaluates whether that STA should be pre-association steered and/or if an existing pre-association steer should be aborted due to insufficient RSSI.

The other role of *bandmon* is to manage the active steering through the use of the projected utilization increase and steering blackout as mentioned above. Other modules, namely, *stamon* and *steeralg*, call the *bandmon* module to determine if steering is currently permitted.

## stamon module

The *stamon* module (short for STA monitor) is the one responsible for starting the non-overload post association steering logic for both idle and active clients. The possible triggers for it to evaluate whether steering should be performed include an RSSI update, a change in the activity status of a STA, a STA becoming eligible for steering again, and a STA becoming eligible for an 802.11k Beacon Measurement again. Whenever one of these triggers occurs, *stamon* evaluates the RSSI and MCS (if relevant), querying the information from *stadb*. It also queries *steerexec* to determine if the STA is even eligible for the type of steering being considered and queries *bandmon* to ensure that type of steering is currently allowed.

Note that in some cases, recent RSSI information may not be available. In this case, *stamon* requests using *wlanif* that a measurement of the uplink RSSI on the serving channel be performed. After a recent RSSI value is available on the current band, *stamon* asks estimator to derive an RSSI value for the non-serving channel.

After all conditions for steering are satisfied, the next step depends on the type of steering being performed. For idle steering, *stamon* calls directly into *steeralg* and requests that idle steering be performed. On the other hand, for active steering, *stamon* asks estimator to measure the STA's data rate on the current channel and the airtime on the candidate channels. At this point, responsibility for the completion of this steering attempt shifts to *steeralg*.

## estimator module

The *estimator* module has two primary responsibilities. The first is to estimate the uplink RSSI on a non-serving channel based on the last measurement on the serving channel. As described above, it does this through the addition of a fixed offset value from one band to the other with an additional adjustment for transmission power differences in the case that the channel being estimated has a lower maximum transmission power than the currently serving channel.

The other responsibility of *estimator* is to estimate the airtime contribution of a given active client on both its current channel and on a candidate channel being considered for steering. It does this through a two step process. In the first step, it enables the byte count statistics and samples them twice, separated by 1 second. These statistics are only enabled when such an estimate is taking

place so as to limit the throughput impact. From these stats, a downlink and uplink throughput are obtained. To arrive at an airtime estimate for the current channel, the last reported downlink rate is first obtained from the driver and scaled by a fixed factor to roughly estimate the MAC layer overhead. This value is then divided into the combined downlink and uplink throughput, thereby resulting in an airtime estimate.

The estimation of a client's airtime on a non-serving channel is done in a similar manner. First, estimator asks the client to perform an 802.11k Beacon Measurement. This results in a downlink RSSI (referred to as an RCPI in the 802.11k terminology). This is then used to estimate an SNR and from that an achievable physical layer rate based on the client and AP capabilities on the target channel. The airtime is then computed in the same manner as that for the serving channel airtime (combined throughput divided by the physical layer rate scaled by a constant factor). For any other channels on that same band, the measured downlink RSSI is adjusted if the maximum Tx power of the AP on that channel is less than the one measured. This adjusted RSSI value is then used in the same manner to derive an airtime value.

Normally *estimator* does these estimates at the request of *stamon* for non-overload steering. However, when a channel is overloaded, it will loop over all of the STAs that are active steering eligible, requesting only that their current channel airtime be estimated. This is then used by *steeralg* (as described below) as the candidate set of STAs with which to perform offloading.

### **steeralg module**

The *steeralg* module is the one that is responsible for ultimately deciding whether post-association steering should be done and if so, what the relative ordering of the candidate BSSes should be (assuming there is more than one). In the case of non-overload steering, each candidate BSS is evaluated to determine if it meets the criteria for steering. All that do are then put into a priority order.

The priority for idle steering depends on whether the optional feature to prioritize based on PHY capabilities is enabled or not. The rules are as follows (in decreasing precedence):

- BSSes that have reserved airtime for the STA are preferred.
- BSSes with utilization below the safety threshold are preferred.
- 5 GHz channels are preferred.
- BSSes with lower measured utilization are preferred.

For active steering, the prioritization is done as follows, with the additional requirement that a BSS is only a candidate if it has sufficient capacity to support the client's projected airtime on that channel without going over the safety threshold.

- BSSes that have reserved airtime for the STA are preferred.
- BSSes with higher available throughput (as computed by multiplying the estimated downlink rate for this STA by the available airtime on the channel) are preferred.

This prioritized list of candidates is then provided to *steerexec* for the actual orchestration of the steering.

*steeralg* is also responsible for coordinating steering under overload conditions. When it receives the *bandmon* event indicating that a channel is overloaded, it first requests estimator to estimate the airtime of each active steering eligible STA on the current channel. Once this completes, it

sorts the STAs according to one of two schemes. The first scheme sorts the clients in decreasing order by their estimated airtime on the current channel. This is the default scheme and attempts to offload the worst offender first. The second scheme considers the client's PHY capabilities first before ordering by airtime. If the overloaded channel is on 5 GHz, 802.11a/n clients will be prioritized ahead of 802.11ac clients. On the other hand, if the overloaded channel is 2.4 GHz, 802.11ac clients will be preferred over other clients. This attempts to place the 802.11ac clients onto 5 GHz and the non-802.11ac clients onto 2.4 GHz if they are contributing to the overload (and other conditions are also met).

Once the prioritized list of candidates is prepared, *steeralg* then requests estimator to determine the first client's airtime on the non-serving channel. Once this is complete (which involves an 802.11k Beacon Measurement as mentioned above), it then applies the above prioritization rules for active steering. If any candidates meet the criteria, the estimated airtime on the current channel for this STA is added to the projected utilization decrease and then the STA is steered.

Once the decision is made for offloading a STA, *steeralg* decides whether to proceed or terminate the offloading process. The process is terminated either when the projected utilization decrease would bring the utilization on the overloaded channel below the safety threshold or a new utilization measurement arrives, thereby starting a steering blackout period. When the steering blackout period ends, if there is still an overloaded channel, offloading will be triggered again.

## Simplifications

For simplicity, even if the steering attempt is rejected by a STA, the projected utilization increase and decrease contributions are still kept. Once the offloading cycle completes, these values will be reset back to 0 to prepare for the next offloading attempt.

## steerexec module

The *steerexec* module is responsible for the actual orchestration of the steering. It offers two primary APIs to request steering be done. The API for pre-association steering takes a set of channels on which to allow the provided STA to associate and blacklists all channels that are not on this list. The API for post-association steering takes a sequence of BSS candidates, ordered from the most preferred to least preferred. *steeralg* and *bandmon* call into *steerexec* with a list of candidate BSSes or channels, respectively. It then makes the necessary calls to *wlanif* to facilitate the client transition. This includes the manipulation of blacklists and/or the sending of the 802.11v BSS Transition Management Request message to the STA being steered.

State is kept for each STA to implement the various compliance, safety, and hysteresis mechanisms mentioned above. This state includes timers, state machines, and various statistics that can be useful in determining how well a given STA reacts to steering attempts.

Once steering completes, *steerexec* is also responsible for determining when any blacklists should be removed. To do so, it maintains the type of steering last performed (channel-based pre-association or BSS candidate-based post-association).

## 16.2.8 Estimation algorithms

### 16.2.8.1 Non-serving uplink RSSI estimation

The equations shown below are how uplink RSSI on a non-serving band/channel are derived from the current uplink RSSI. Here the TxPowerDelta is the maximum Tx power of the serving channel subtracted from the maximum Tx power of the non-serving channel.

Equation for Estimating 5 GHz RSSI from 2.4 GHz RSSI

```
RSSI_5 = RSSI_24 + RSSIDiff_EstW5FromW2 + min(0, TxPowerDelta)
RSSI_5 = RSSI_24 - 15 + min(0, TxPowerDelta)
```

Equation for Estimating 2.4 GHz RSSI from 5 GHz RSSI

```
RSSI_24 = RSSI_5 + RSSIDiff_EstW2FromW5 + min(0, TxPowerDelta)
RSSI_24 = RSSI_5 + 5 + min(0, TxPowerDelta)
```

## 16.2.9 Disable blacklist functionality in band steering

This will remove the blacklist functionality from band steering and is a configurable option.

### 16.2.9.1 LBD changes

- To enable or disable this feature a configuration parameter “AuthAllow” is introduced in the lbd config file.
- When this feature is enabled, a new ioctl is sent to the driver to set a new flag across the MAC address present in the ACL list in the driver.
- This ioctl is sent along with the ioctl to set the probe response withholding flag.
- The ioctl to clear the Auth Allow flag is also sent when the probe response withholding flag is cleared.
- When the event is received from the driver then the print indicates that the authentication is allowed due to the setting of this flag.

### 16.2.9.2 Driver changes

- In the driver a new flag is added in the ACL entry and when the flag is set, it allows authentication response.
- This flag will be set/cleared whenever the lbd sends an ioctl to the driver for the particular MAC address.
- A wifitool command is also present to set/clear flag for the MAC address present in the ACL list.
- When there is authentication request for the particular MAC for which the AUTH ALLOW flag is set then authentication is allowed.
- When authentication is allowed due to the setting of the flag then an event is sent to the lbd which prints the corresponding message.

### 16.2.9.3 Assumptions

- Enabling this feature might bring down the steering success rate.
- Steering might fail for the clients who sends an AUTH directly on previously serving band when being steered. The steering success rate characterization and tuning is left to the customer.
- When this feature is enabled using the configuration parameter, then it is applicable for both the bands. In other words, this is a global feature that is applicable across all Radios and cannot be enabled/disabled on individual radios.
- The T\_Steering timer logic is not affected i.e. the timer starts, whenever the first authentication rejection happens.
- This feature can be enabled/disabled only through lbd and not though command line (via iwpriv or wifitool)
- The lbd daemon is restarted whenever this configuration parameter is changed so that the latest changes becomes effective.

## 16.3 Multi-AP coordinated steering and adaptive path selection

In a multi-AP environment where there is a Central AP (CAP) and one or more range extenders (REs) connected to the CAP over Wi-Fi, some additional factors must be considered to manage the overall network load so as to make better use of the available resources. A number of conditions can occur that can lead to reduced performance, including:

- Clients that stay connected to one access point even though they would be better off served by another access point. This could be both scenarios where the STA maintains its association with the CAP even though its signal is weak and it would be better off served by the RE, and scenarios where it stays associated with an RE due to a strong signal even though its end-to-end performance impact to the network would be improved by associating to the CAP (due to the elimination of two Wi-Fi hops).
- Reduced performance for STAs connected to range extenders due to a fixed selection of backhaul path that does not adequately consider the differing bandwidths available on 2.4 GHz and 5 GHz.

The first issue can be solved by introducing an AP steering mechanism (in addition to the single AP band steering described in [Section 16.2](#)) to help the client move to a new access point at the right time. The second issue can be addressed by introducing more sophisticated bridging rules on the CAP and range extenders in order to dynamically adapt to the link conditions and load in the network. Each of these is described in a section below.

These features are part of the larger Wi-Fi Self-Organizing Network (Wi-Fi SON) umbrella to enable better user experience within the home.

## 16.3.1 Multi-AP coordinated steering

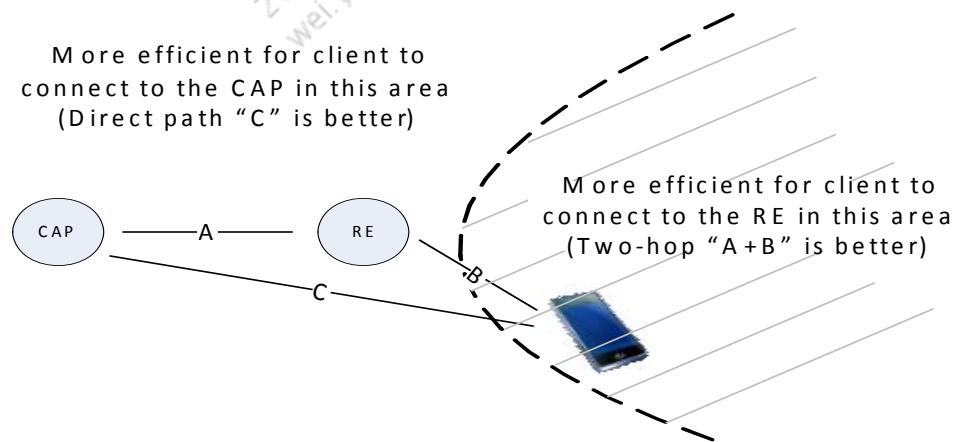
### 16.3.1.1 Design overview

There are two aspects to multi-AP steering that need to be considered. The first is defining the points at which steering between APs may be appropriate and how to determine that it is indeed warranted. The second is how to orchestrate the steering itself and ensure there is not too much churn in the network. Each of these aspects is considered in turn below.

Broadly speaking, in a star topology (as assumed in the current implementation) with a Central AP and one or more Range Extenders connected to the CAP via Wi-Fi, the point at which AP steering should be considered is dependent on which AP is serving a particular STA. This is because it is important to not only consider the signal strength of the STA to its serving AP but also the path that its traffic must take through the network.

Since the majority of client traffic within a home environment is either destined to the Internet or to devices connected to the CAP, optimizing only for the STA signal strength can lead to reduced performance if a STA is moved to the range extender prematurely. Such movement causes reduced performance due to the additional Wi-Fi hop incurred when the range extender forwards data to the CAP or vice versa. On the other hand, a STA that is close to the range extender may be better off served by the CAP despite the signal strength of the range extender still being more than adequate from the STA's perspective.

The diagram below shows a pictorial representation of this tradeoff. Here, when the client is in the shaded region, it is best served by the range extender. However, when it is in the unshaded region, it is best served by the CAP even though it may see the range extender signal strength as strong.



Evaluating whether a client has crossed between these regions is the responsibility of the serving AP. The serving AP monitors the client's uplink RSSI (as measured either by uplink data packets sent by an active client, uplink QoS null packets sent periodically by an idle client, or the signal strength of ACK packets sent by the client in response to a QoS Null Data Packet sent by the AP when the client is first detected as idle). The serving AP, knowing whether it is a CAP or an RE based on configuration parameters, uses the appropriate threshold. On the CAP, the threshold is lower to delay AP steering until the STA has likely crossed into the gray region. On the RE, the

threshold is higher to cause the next stage of the process to take place when the client is close to the RE.

After the uplink RSSI threshold is crossed, the next step is for the serving AP to request the STA to perform an 802.11k Beacon Measurement in order to determine whether there actually is another AP that is a better candidate for the STA. This is because it is not possible based on uplink RSSI alone on the serving AP to tell whether the STA is moving closer to another AP in the network or is moving farther from all APs. The 802.11k measurement solves this ambiguity by allowing the serving AP to compare the signal strength of itself against that of other APs in the same ESS that can be seen by the client. In the default configuration, the serving AP will steer the STA to a different AP only if that other AP has a downlink signal strength a certain amount greater than that of the serving AP. The exact value used is dependent on whether the STA is being steered towards the CAP, towards a range extender, or between range extenders. Note that although the implementation supports configurations where the target AP is allowed to be weaker than the serving one, this is not used in the default configuration as some clients will reject such steering attempts.

Since AP steering decisions are based on 802.11k measurements, clients that do not support this standard will not be candidates for AP steering (but can still be band steered). Movement of such STAs between CAP and RE will solely be subject to their own roaming algorithms. However, since in this multi-AP coordinated steering mode all APs have the same SSID, most STAs should at least roam on their own when the signal strength of the serving AP becomes weak.

Once a determination is made to steer a STA, the appropriate over-the-air mechanism is used based on the STA's capabilities and current steering behavior classification. The mechanics of the steering operation in a multi-AP network are more than that used in a single AP environment. For the best probability of steering success, it is necessary that the node initiating the steering coordinate with the other nodes in the network so that they can install blacklists and withhold probe responses as appropriate.

Without this coordination, the legacy steering mechanism would often lead to the client associating on a different BSS than intended in cases where signal strength is similar between CAP and RE or the client's own BSS selection algorithm prefers a different BSS than the target one. Even steering using 802.11v BSS Transition Management Request (at least with current clients) is not as reliable without the steering coordination.

The coordination itself is done through vendor specific messages sent using IEEE1905.1 format. The messages related to steering itself are as follows:

- Prepare for Steering Request/Response - The request is sent to all nodes in the network with the target BSS information and the type of steer being performed. In addition, the state of the client with respect to the various types of steering is also sent to all nodes, along with the pollution time remaining on each channel local to the node sending the Prepare For Steering Request (with 0 meaning no pollution). Each node examines its own information about the STA (if it has any) and accepts the request if its state indicates the client is not prohibited for the type of steering being performed. It also records the pollution information in its local database to avoid steering back to a BSS that is considered polluted. Once responses have been collected from all nodes in the network, the initiating node performs the steering of the STA (using either the legacy mechanism or 802.11v BSS Transition Management Request) if all responses were successful or aborts the steering if any nodes rejected the request.

- Abort Steering Request/Response – The request is sent to all nodes in the network whenever any node in the network needs to abort an in progress steering attempt. The reasons for an abort are numerous but generally consist of either: one of the steering safety mechanisms (as defined in [Section 16.2.4](#)) being triggered or one of the nodes rejecting the steering attempt (due to having state information locally that precludes the STA from being steered). When a node receives such an abort, it removes the blacklists and cancels its expectation that the STA will associate on the target BSS. After doing so, it sends the response.
- Authentication Reject Sent – This message is sent by a node whenever it sends an Authentication message with the refused code to a STA due to an attempt by the STA to authenticate on a blacklisted BSS. This allows the serving AP to use the appropriate prohibit timer and also monitor authentication rejects across other APs in the network for use in the steering safety mechanism.
- STA Info Request/Response – This request message allows an AP that believes it does not have complete or up to date information about a STA to request it from other nodes in the network. The typical scenario where this is used is when a STA associates to an AP without being steered there. In this case, the newly serving AP may not know the capabilities of the STA on the non-serving band and may also not know its current steering state. By sending this request, all other APs can send their most recent info and the newly serving AP can take the union of the information (using the most pessimistic values received for the steering state). The STA Info Response contains the local pollution info, client capabilities, and the state of the client with respect to the various types of steering.

In addition to the messages to orchestrate the steering itself, there is another aspect of the coordinated steering implementation that is different from single AP steering. Specifically, active offload steering (where STAs are steered away from an overloaded band) and active upgrade steering (where STAs are steered from 2.4 GHz to 5 GHz) are only allowed when the AP is assigned a steering slot by the CAP.

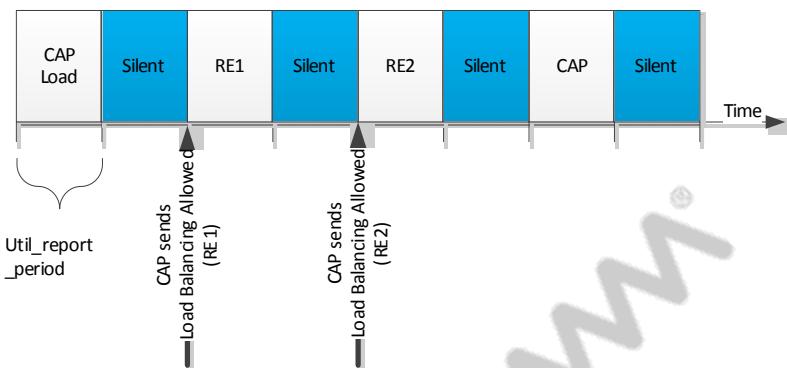
These slots are preceded by the collection of average utilization information from all nodes. This information is then shared with all APs at the same time one AP is being assigned a steering slot. The CAP will round robin through the nodes in the network (including itself), assigning a steering slot one at a time.

During its steering slot, the assigned AP can only perform one active offload or upgrade steer. This limitation is imposed because in a multi-AP network, it is harder to predict the impact of such a steer due to the possibility it affects the path used on the backhaul.

After the AP performs such a steer or if it determines it has no STAs that would be eligible for such a steer, it informs the CAP that it is complete performing load balancing. If a STA was steered, then the CAP will institute a steering blackout at the beginning of the next slot. A new AP will be assigned a steering slot only after this steering blackout period expires. The use of a blackout period ensures that the impact of steering can be measured for the duration of the medium utilization averaging window (which defaults to 30 seconds).

If the AP assigned the load balancing slot does not steer any STAs and responds in the first half (by default) of the steering slot, the CAP will assign the next node in the sequence the remainder of the slot. This can speed up the cycle in the case that active steering eligible STAs are only associated on a subset of the APs.

The diagram below shows an example of the CAP assigning steering slots to different nodes in a 3 AP network. Here it is assumed that each node performed an active steer during its assigned slot and thus a steering blackout follows its slot.



The messages that implement this slotted steering scheme are the following:

- Average Utilization Request – Sent periodically by the CAP to each RE, asking them to report back the average utilization measured for each of the channels on which they are operating.
- Average Utilization Report (single) – Sent by an RE back to the CAP with the average utilization measured on each operating channel since the last Average Utilization Request.
- Average Utilization Report (aggregate) – Sent by the CAP to all REs to convey the maximum utilization measured by all nodes in the network over the last measurement window. This ensures all nodes know the worst case utilization.
- Load Balancing Allowed – Sent by the CAP to a single RE to indicate that it can now perform active offloading and active upgrade steering. This message also conveys the maximum utilization values in case the aggregate Average Utilization Report is lost.
- Load Balancing Complete – Sent by an RE back to the CAP to indicate whether it performed any active steers during its steering slot.

**NOTE** Some forms of steering are still allowed even when the node is not assigned a steering slot. Specifically, idle steering is allowed as the expectation is that it will have limited to no impact on medium utilization (as a STA that is currently idle is more likely than not to continue to be idle after the steer). Active downgrade steering, AP steering and Interference Avoidance Steering (IAS) are also allowed, as these are done to ensure the client does not lose its Wi-Fi connection abruptly.

Finally, it is important that all nodes in the network have the same steering configuration parameters (those described in [Section 16.2](#)) so that they have a consistent view of the steering state of each STA. Currently there is no messaging that ensures this so care must be taken when tweaking configuration parameters that the same change is done on all nodes.

### 16.3.1.2 Implementation details

The bulk of the logic to implement the multi-AP coordinated steering leverages the code developed for the single AP steering. To support reuse of this code, the code that is applicable in

both the single AP and multi-AP case is built into a shared library named **liblbcmlibs.so**. In most of the modules, specific extension points are defined where an implementation for single AP steering can be provided within the **lbd** source tree and an implementation for multi-AP steering can be provided within the **hyd** source tree. All of these multi-AP implementation aspects are contained within the **w1b** directory of the **hyd** source tree. The following are the multi-AP aspects for each of the modules:

- *bandmon* – Implements the round robin assignment of the steering slot on the CAP and the logic to wait for such an assignment on an RE. Also responsible for averaging the medium utilization reports received from *pcwService* and providing these when requested in response to Average Utilization Request.
- *estimator* – Adds new logic for handling 802.11k Beacon Reports containing BSSes of remote nodes. Specifically, it uses the topology database (populated based on some vendor-specific TLVs in the IEEE1905.1 topology messaging) to resolve the physical layer capabilities (number of spatial streams, standard, and transmission power) of remote BSSes so that this can be used when estimating rate and airtime. The multi-AP version of this module also adds estimation of same band BSSes that are not included in the report (such as those operating on a different channel than that measured).
- *stamon* – The multi-AP specific portion of this module determines whether AP steering criteria have been met or not when an updated RSSI is available. If the AP steering requirements have not been met, then the normal upgrade/downgrade steering logic from the single AP code is used.
- *steeralg* – This module contains the logic to select the best AP for a client based on an 802.11k Beacon Report and then from that, determine the best channel to which to steer the client. When the current AP is the best one, it will use the same logic as in the single AP case. This module also has new logic to select which channel is used for an 802.11k Beacon Measurement depending on the type of steering, with 5 GHz preferred for an AP steering trigger and the single AP rules being used in other cases.
- *steerexec* – This module contains the necessary integration with the steering coordination messaging. It is responsible for triggering the send of all steering-related messages and handling any response messages received.
- *wlanif* – This module contains additional logic to be able to resolve BSSIDs to/from their internal representation even when those BSSIDs are on remote nodes. In the remote node case, the resolution is done using information in the topology database (as populated by IEEE1905.1 topology messaging). This module also ensures the Wi-Fi Self-Organizing Network (SON) mode capability is set in the IE that is used for range extender automatic mode switching as described in Section

In addition to the multi-AP versions of the existing modules, there is one new module named *steermsg*. This module contains the code to send and receive the steering coordination messages described above. It also is responsible for updating information in the station database upon receipt of a STA Info Response (or similar information contained within another message).

### 16.3.1.3 Available debugging information

The debug CLIs for the modules implementing multi-AP coordinated steering are unchanged from their single AP implementations. However, when performing AP steering manually (using *steerexec steer*), one must know the AP ID to specify as the first argument after the MAC

address. If there are only two APs in the network (CAP and one RE), it is generally safe to assume the other AP's ID will be 0. On the other hand, in a multi-AP network, the remote APs can be detected in either order from run to run. To determine which AP is which, use the `td s` debug CLI command. An example of this output is shown below.

```
@ td s
Topology Discovery Service module status:
Mode of operation: Relaying device
IEEE 1905.1 mode: QCA Enhanced
Hy-Fi 1.0 compatibility: Supported
-- ME:
    QCA IEEE 1905.1 device: 00:03:7F:15:F9:75, IPv4 address: 192.168.1.1
    Local interfaces:
        Interface name      Medium Type   MAC Address   Contention   PHY
    Capabilities
        ath0                WLAN5G       00:34:56:78:1D:1D 149
    80MHz,4,VHT,9,20
        ath1                WLAN2G       8C:FD:F0:00:E4:93 11
    20MHz,4,HT,7,28
        eth1                ETHER        00:03:7F:15:F9:75 255

    Legacy Devices:
        Interface eth1:
            00:02:B3:3F:DD:C3

-- DB (1 entries):
    #1: QCA IEEE 1905.1 device: 00:03:7F:3F:04:4B, IPv4 address:
    192.168.1.191 (message ID: 00107; ts: 1436466636)

    Remote connections (Directly connected to self):
    Interface name      Medium Type   MAC Address   MID      Direct   Legacy
    Bridge   TS          Contention   PHY Capabilities
        ath0                WLAN5G       06:34:56:78:D8:D8 00107   Yes     No
    1436466623 149
        ath1                WLAN2G       92:FD:F0:00:E1:F2 00107   Yes     No
    1436466623 11

    Remote connections (Not directly connected to self):
    Index      Medium Type   MAC Address   Contention   PHY Capabilities
        0        WLAN5G       00:34:56:78:D8:D8 149        80MHz,4,VHT,9,28
        1        WLAN2G       8C:FD:F0:00:E1:F2 11         20MHz,4,HT,7,28
        3        ETHER        00:03:7F:3F:04:4B 255
        4        ETHER        00:03:7F:85:D5:75 255

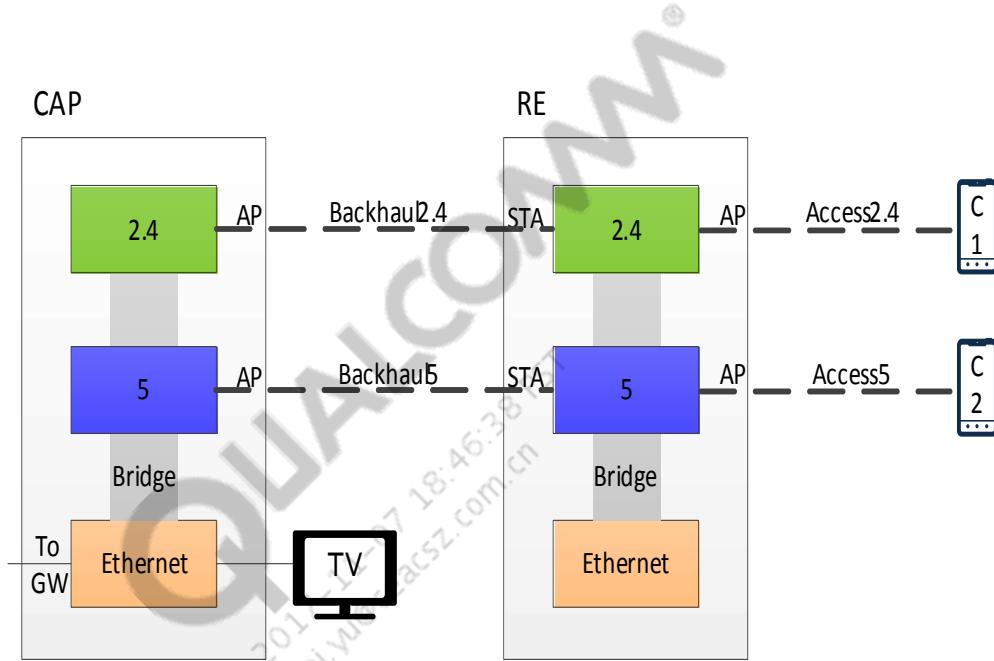
    0 Bridged addresses:
    message ID: 00000, time stamp: 1436466636
    Address      Interface   Interface Type
```

The main aspect to note in this output is the number assigned to each entry in the DB section. To use this as an AP identifier when steering, subtract one from the number.

## 16.3.2 Adaptive path selection

### 16.3.2.1 Design overview

Many range extenders in the market today are Dual Band Dual Concurrent and connect back to the primary AP (referred to as the Central AP or CAP here) on both bands. Such a network can be viewed as shown in the following figure as having a backhaul interface on each band and an AP interface (called an access interface) on each band providing connectivity to nearby STAs.



Traditionally these DBDC range extenders implement a fixed selection of the interface to use when sending traffic upstream. This is generally done to avoid possible bridging loops in the network that can lead to a broadcast storm. Such fixed forwarding schemes can also work with CAPs that act purely as learning bridges, as the CAP will learn the MAC addresses of the RE and clients behind the RE (whether wired or wireless) on a single interface. This means the uplink and downlink paths are the same and are controlled by how the RE selects the uplink path.

Two fixed forwarding models are generally used, either forwarding traffic to the backhaul interface on the same channel as that used by the STA associated to the RE or forwarding traffic to the interface on the other band. Each of these schemes has advantages and disadvantages. The cross band approach eliminates the impact of self-contention between the access link and backhaul link which can sometimes lead to better performance. However, due to the differing bandwidths on 2.4 GHz and 5 GHz (for example, typically 20 MHz and 80 MHz), such cross band forwarding can lead to the 2.4 GHz backhaul interface becoming the bottleneck for clients associated on 5 GHz. On the other hand, same band bridging can lead to under-utilization of the other band. Depending on the distribution of the clients across the bands, this can lead to significantly reduced performance for all clients.

Adaptive Path Selection (APS) allows for different forwarding paths to be taken per destination or even on a per flow (with some limitations noted below) when both the CAP and RE implement this algorithm. There are three core aspects to this algorithm, namely loop avoidance, topology and

link metric collection, and the actual path selection and load balancing. Each of these will be considered in turn.

### Loop avoidance

As mentioned above, when multiple interfaces are connected to the same upstream device, it is important to ensure that packets do not get forwarded in an endless loop. Traditional spanning tree protocols can be used to avoid loops in bridged topologies, but they lead to one of the upstream facing interfaces not being used until the primary one (termed the root port) fails. Because the conditions on a wireless network are not as predictable as they are on a wired network, being restricted to only a single interface can be sub-optimal.

APS solves this problem by introducing the principle of upstream and downstream facing ports, referred to as non-relaying and relaying ports in the implementation. For the upstream facing ports, no bridging is allowed between them to avoid looping. Additionally, only one upstream port is allowed to transmit broadcast traffic between the downstream ports in order to avoid seeing broadcast frames twice. On the other hand, no restrictions are applied to downstream facing ports. Frames can be forwarded between any of them.

Determining whether a port must be a downstream or upstream facing port is currently based on heuristics. STA interfaces are always considered upstream-facing and therefore, these interfaces are placed in the non-relaying group. AP interfaces and Ethernet interfaces are considered downstream-facing and therefore, these interfaces are placed in the relaying group. This handles topologies where the RE is acting as a wireless range extender. When an RE device is connected to the CAP via Ethernet or via a separate PLC network, these heuristics will lead to a bridging loop. However, if this feature is used in conjunction with the RE Placement and Auto-Configuration feature, such topologies can be supported (provided the device is booted with the Ethernet connection) as the STA interfaces are disabled.

### Topology and link metric collection

In order for the path selection algorithm to make decisions about how to forward traffic, it needs to collect information about the topology of the network and the strength of the links in the network. The network topology is learned through the use of IEEE1905.1 Topology Discovery, Query, Response, and Notification messages along with some vendor specific extensions to these. Discovery messages enable nodes to determine which of their interfaces are connected to other IEEE1905.1 devices, whereas the Query and Response messages allow nodes to get the complete topology information from a remote node. This information includes the remote interfaces (those to which this node is not directly connected) and the MAC addresses learned on each interface.

In addition to the information provided by IEEE1905.1, additional information is provided that includes the list of associated STAs (to help in disambiguating where a given MAC address in the network is actually located) and Wi-Fi capability information for other AP interfaces. Nodes also include additional information in the Topology Notification message when a STA associates locally to assist other nodes in quickly updating their bridging information, as normal Topology Notifications do not include any information and require the node to query the sending node to determine what actually changed.

Link metric information is the other key element of making intelligent forwarding decisions. IEEE1905.1 defines a means by which one node can query another node for information about the physical and MAC layer characteristics of its links. However, as the standard does not clearly

define how these metrics should be measured or interpreted, additional vendor specific metrics are included in the Link Metric Query/Response messages. These vendor specific metrics consist of capacity values for both TCP and UDP traffic as estimated by the sending node. It is these values that are used in determining the path to take for a given client.

Furthermore, the standard only defines link metrics between IEEE1905.1 devices. As APS makes decisions based on information about the final hop (to an associated STA), additional link metric information is exchanged by nodes with this APS feature. These nodes also periodically send dummy frames to one another to ensure they can get an updated rate value from the Wi-Fi driver and firmware, as this is used in the capacity calculation described below.

For each type of traffic (TCP and UDP), nodes estimate two capacity values. The UDP full capacity is an estimate of the application layer data rate achievable assuming a completely idle medium. This value is derived from the last physical layer data rate chosen by the rate control algorithm. This last transmit rate is compared against two thresholds to determine which of three regions it falls into. Different scaling factors (based on simulations of effective data rates for given physical layer rates) are applied to arrive at an actual full capacity value. A further scaling factor is applied to arrive at a full capacity value for TCP. This scaling factor estimates the impact of additional protocol overheads incurred when using TCP. These full capacity values are then further adjusted based on packet error rates reported by the lower layers.

After deriving the full capacity value, the available capacity is determined by scaling the full capacity by the remaining time available on the medium (up to a configurable maximum utilization that is less than 100%). It is this value that is used in the path selection decisions.

### **Path selection and load balancing**

Adaptive Path Selection operates by combining the topology and link metric information to arrive at the best path to be taken for a specific destination or flow. At a high level, it contains two processes, one to pick the default paths in the network (adapting as loading conditions change) and one to change paths of existing flows in the network during overload.

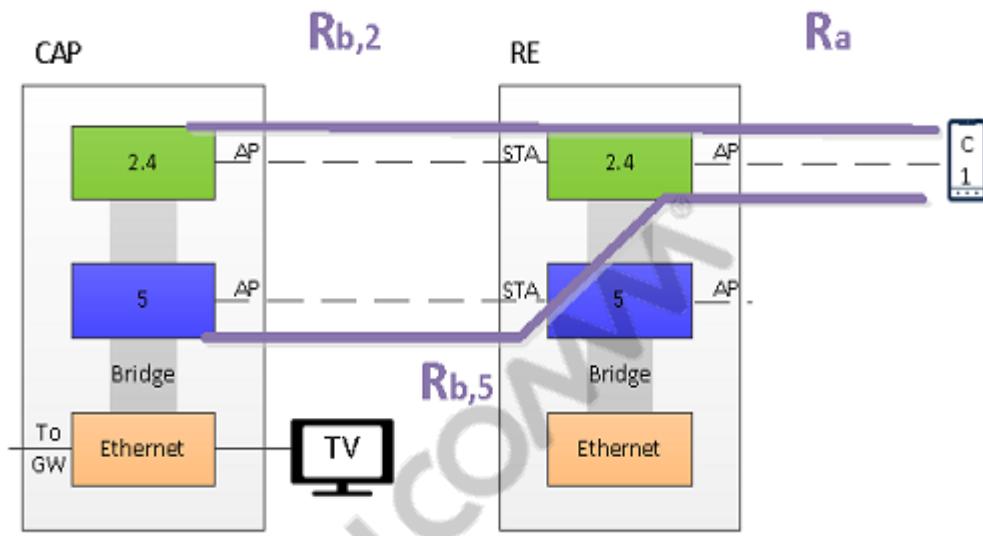
The default path selection process runs on each node and determines the best egress interface for all MAC addresses that are connected to other nodes in the network. This determination is done both on a periodic basis as well as when significant events occur. Such significant events can include changes in topology and large changes in medium utilization or the physical layer rate on a link to other nodes in the network.

When determining the default egress interface for a given MAC address, an end-to-end path is computed for each possible path. When the destination MAC address is not a Wi-Fi STA (for example, it is an AP device itself or a device connected to such a device over Ethernet), APS picks the interface with the best available capacity.

When the destination MAC address is a STA associated to a neighboring AP, it estimates the end-to-end data rate for all possible backhaul paths to reach the STA. When this path involves the use of the same channel for both the backhaul link and the access link, the impact of self-contention is estimated by multiplying the two available capacities and then dividing that by their sum. When the path involves different channels for the backhaul and access links, the minimum of the available capacities is used as the end-to-end path.

For example, in the topology shown in the figure below, the end-to-end data rate for the path that uses 2.4 GHz on both the backhaul and access links will be estimated as  $(R_{b,2} \times R_a) / (R_{b,2} + R_a)$ .

$+ R_a$ ) whereas the path that uses 5 GHz on the backhaul link and 2.4 GHz on the access link will be estimated as  $\min(R_b, 5, R_a)$ .



One other special case is determining the path from one RE (RE1) to another RE (RE2) or a device associated to RE2. Since the CAP will perform intra-BSS bridging, APS will consider there to be a virtual link from RE1 to RE2 on each channel they share. The self-contention calculation used above will be applied to this virtual link. Any further contention impact will be applied for the final hop to a Wi-Fi STA associated to RE2.

The second process in Adaptive Path Selection is the load balancing procedure that can move active flows to a new interface. Flows are identified by their destination MAC address and an 8-bit hash over certain fields in the packet (typically source and destination addresses and port numbers). As the hash is limited in size, it is possible for two distinct flows to map to the same hash. Note however that if they are destined to two distinct MAC addresses, they will still be treated as separate flows.

This flow movement process can be triggered by changes in the topology of the network and when the medium utilization on a channel rises above a configurable value (70% by default). For topology changes, the algorithm looks for a new interface based on the available capacity. Reacting to a medium over subscription is a multi-step process. First, all flows on the oversubscribed interface are ranked according to their contribution to the medium utilization. Then, starting from the highest contributor, each interface that has a link to the destination is examined. If an interface is found that has a configurable amount (10% by default) more capacity than the flow rate, then that interface is selected. If no interfaces can be found, the next highest flow is examined. This process continues until a flow is moved or a certain number of flows have been examined with none being a candidate.

After a flow is moved due to oversubscription, any further load balancing is delayed until updated medium utilization information is available. This behavior prevents too many flows from being moved because the impact of such flows to utilization is not fully reflected at that point.

**NOTE** The path switching for flows is done locally on the AP through which the flow is passing. It is transparent to the actual end-client.

**NOTE** While adaptive path selection (APS) decisions are taken, it is not possible to accommodate utilization higher than the safety threshold on target bands. This problem is not seen with TCP and it is noticed only when UDP traffic of bandwidth more than 90 Mbps (when 2 GHz band is oversubscribed). As a result, it is necessary to restart flows if they must be moved to a different interface.

### 16.3.2.2 Implementation details

The implementation of Adaptive Path Selection is split into two components—one component resides in the control path, whereas the other component is located in the data path.

#### Control Path – hyd

The control path portion is responsible for configuring the bridge and selecting the paths based on topology and link metric changes. The **hyd** daemon contains the control path implementation. The key modules implementing path selection are as follows:

- *tdService* – This module is responsible for learning the topology of the network based on IEEE1905.1 messaging.
- *pcwService* – This module handles link metric information from the driver for all Wi-Fi interfaces. It translates the raw physical layer metrics into capacity values and makes these available to other modules. It also has the logic to ensure that medium utilization information is updated after an APS load balancing operation takes place.
- *psService* – This is the core module that implements the path-selection algorithm described above. It reacts to events from other modules indicating a change in topology or link metrics and reprograms the bridging tables accordingly.
- *heService* – This module collects periodic byte count samples for active flows in the network and estimates their rates. This information is used by *psService* while determining the manner in which flows must be moved from one interface to another.

The preceding four modules are the primary ones relevant for APS. Apart from these primary modules, a few other support modules within **hyd**,

#### Data Path – Bridging infrastructure

The Adaptive Path Selection data path is implemented using Linux kernel Net filter hooks that are attached to the bridge for certain operations (such as forwarding and local input). These hooks implement the three key aspects of APS in the data path:

- Preventing bridging loops – The forwarding and flooding hooks prevent packets received on an upstream (non-relaying) interface from being forwarded on another upstream interface. They also ensure only a single broadcast frame is passed through to the downstream (relaying) interfaces.
- Default path selection control – A bridging table, named *H-Default*, can be programmed by user space to override the bridging that would occur by using the Forwarding Database (FDB) of the bridge itself. This table consists of destination MAC addresses and the name of the interface to use for UDP and non-UDP traffic.
- Active path selection control – A bridging table, named *H-Active*, enables the control of the interface used for an individual flow. An entry in this table is created when an entry in *H-*

*Default* or FDB is matched. The entry tracks the final destination address, the hash of the flow, statistics about the flow, and the egress interface (along with some additional information not covered here). An entry can be written by user space to force that flow to use a different egress interface.

These hooks are activated when **hyd** attaches to the bridge. When active, the process of selecting the egress interface for a given destination address (DA) is as follows:

- Compute the hash for the flow based on the fields in the packet and look up in *H-Active* to determine if that (DA, hash) pair is present. If so, update the stats and use the egress interface selected. Otherwise, proceed to the next step.
- Look up the DA in the *H-Default* table. If a match is found, use the egress interface indicated based on whether the packet is UDP or non-UDP.
- Use the normal FDB-based forwarding rules when there is an entry in the FDB. If no entry is found, flood the packet on all downstream (relaying) ports and the one designated broadcast upstream (non-relaying) port, excluding the interface on which the packet was received. This logic also applies for broadcast/multicast traffic.

### 16.3.2.3 Available debugging information

The debug CLI provided by **hyd** can be useful in displaying both the inputs to and outputs from the Adaptive Path Selection algorithm. The sections below provide some helpful commands that may be used.

#### Topology and link metric information

The **td s** command (an example of which is shown in [Section 16.3.1.3](#)) is useful in confirming all links are detected properly. It also shows each of the MAC addresses that remote devices are detecting on their interfaces. This is the information used by the path selection module when programming the *H-Default* table.

Link metric information can be displayed using the **pc s 2** and **pc s 5** commands, the former of which is for 2.4 GHz and the latter of which is for 5 GHz. Example output is shown below:

```
@ pc s 5
pcService module status:
Status for Wi-Fi interface ath0, 5GHz:
    Debug mode: off
    Timer: Remaining time 0.655972 sec
    AP medium information: Medium characterized/measured at 1441052577 (1
seconds ago)
    Medium utilization = 0%
    Reported medium utilization = 0%
    RAW DATA: Tx Queue Buffer Usage = 0
    RAW DATA: Current Load by node itself = 0
    Number of links = 2
Link #0: DA = 06:34:56:78:09:09 --> Available/Full TCP Link capacity =
671/959 Mbps, Available/Full UDP Link capacity = 828/1184 Mbps
    RAW link data:
        --> raw phy rate = 1559
        --> average aggregation = 96
        --> PHY Error Rate = 0
```

```

--> Last PER = 0
--> MSDU Size = 1300
--> Derived raw TCP Estimated Throughput = 959 Mbps
--> Derived raw UDP Estimated Throughput = 1184 Mbps
Link #1: DA = FC:C2:DE:25:8D:DA --> Available/Full TCP Link capacity =
399/571 Mbps, Available/Full UDP Link capacity = 470/672 Mbps
RAW link data:
--> raw phy rate = 779
--> average aggregation = 96
--> PHY Error Rate = 0
--> Last PER = 0
--> MSDU Size = 1300
--> Derived raw TCP Estimated Throughput = 571 Mbps
--> Derived raw UDP Estimated Throughput = 672 Mbps
Remote STA medium information (1 BSSIDs known):
STAs associated with 8C:FD:F0:00:E4:93 :
Received at 1441052577 (1 seconds ago) (LastMessageID = 586,
LastFragmentID = 0)
AP medium utilization = 20%
AP number of links = 1
AP link #0: DA = FC:C2:DE:26:04:DD --> Available/Full TCP Link capacity =
56/112 Mbps, Available/Full UDP Link capacity = 62/125 Mbps

```

This example is taken from a device that only has AP interfaces. The first portion of this dump displays the information measured locally on this device, including some of the metrics from which the capacity values are derived. The last section (Remote STA medium information) shows metric information for a remote AP interface to which this node is not directly connected. Note that due to implementation considerations, this information can be displayed multiple times in certain circumstances.

Similar information is dumped when performing this command on a device with both STA and AP interfaces. One difference in the output is that it will contain not only the information estimated locally but also that reported by the AP to which the STA interface of the RE is associated. This information allows the RE to know the link rate for other STAs within this BSS and is used in path selection computations.

## Bridging information

To display the current configuration of the bridge (in terms of which ports are upstream and which are downstream), use the `hyctl show` command, an example of which is shown below:

```

root@OpenWrt:/# hyctl show

  br_name br_mode tcp_sp   interfaces      type    grp_num bcast  port_type
  br-lan   GRPREL disable ath11           norelay 0      enable  Wi-Fi 5GHz
                           ath1             relay   1      enable  Wi-Fi 5GHz
                           ath01            norelay 0      disable Wi-Fi 2GHz
                           ath0              relay   1      enable  Wi-Fi 2GHz
                           eth1             relay   1      enable  Ethernet
                           eth0             relay   1      disable Ethernet

```

Here the `ath01` and `ath11` interfaces are STA interfaces that are facing upstream, with `ath11` being the one selected for broadcast/multicast/unknown unicast traffic. The other interfaces are downstream facing interfaces.

To display the default path information, use the `hy hd` command as shown below.

```
@ hy hd
H-Default table contains 3 entries
index      MAC ID          UDPPort      OtherPort      Static
0:        68:05:CA:05:0C:C1  00:03:7F:3E:79:63  (30) ath11 (30) ath11  0
1:        8C:FD:F0:01:41:22  00:03:7F:3E:79:63  (30) ath11 (30) ath11  0
2:        00:03:7F:3E:79:63  00:03:7F:3E:79:63  (30) ath11 (30) ath11  0
```

This shows three entries, all of which whose default egress interface is `ath11` (which in this example is the 5 GHz STA interface of a range extender). The `ID` column indicates the next hop AP device through which traffic will be forwarded. Typically the UDP and non-UDP interfaces will be the same, although they may differ depending on configuration and link conditions.

To display the active path information, use the `he s` command, an example of which is shown below:

```
@ he s 1K
Reporting all active flows with rate > 1000 bps
HSPEC Estimation Status: OK
Current active flows:
Index Hash  MAC ID Rate Port  Idx Age CreateTime NumPackets NumBytes
Priority Serial Flags
[000] 0x71 00:02:B3:3F:DD:C3 00:03:7F:15:F9:75 1020 eth1 (03)
0 4294779 2 106 0x80000000 0 O-S--
Per interface flow rate summary:
eth1: 1020bps (0.00Mbps)
Total: 1020bps (0.00Mbps)
```

In this example, only flows greater than 1 Kbps are being displayed. The destination address is shown in the MAC column and its currently estimated rate is also shown (in bps). After all flows matching the rate requirements are displayed, a summary of the traffic on each interface is shown.

## 16.4 Suspension of probes in 2.4 GHz band

On top of single AP band steering, an add-on functionality to enable suspension of probes in 2.4 GHz band is introduced. This feature allows user to configure the AP such that probe responses are suppressed on 2.4 GHz band, which effectively forces a dual band client to select 5 GHz band at association time. By default, when a dual band client tries to associate with an AP, client scans on both the bands 2.4 GHz and 5 GHz. Therefore, it is possible for a client to connect initially with 2.4 GHz band and then be pushed to 5 GHz band based on band-steering decisions.

To prevent such a scenario, the user can configure the Load Balancing Daemon (lbd) for suppressing the 2.4 GHz probe responses from APs for minimal time and probe request count. This feature is enabled only if the received probe request RSSI of the client is greater than configured threshold. These parameters are configured in lbd configuration file under the ‘steerexec\_adv’ section. The following are the parameters in the lbd configuration file:

| Parameter                  | Default value | Description  |
|----------------------------|---------------|--|
| Delay24GProbeRSSIThreshold | 35            | The minimum RSSI threshold to delay probe responses in 2.4 GHz band (in dB). |

|                          |   |   |
|--------------------------|---|---|
| Delay24GProbeTimeWindow  | 0 | The time window within which probe responses will not be sent for configured count(s).                |
| Delay24GProbeMinReqCount | 0 | The probe request count above which probe responses will be sent for all received within time window. |

By default, this feature is disabled and it can be enabled using configuring the aforementioned parameters with values other than zero. The probe responses are suppressed for Delay24GProbeMinReqCount times in Delay24GProbeTimeWindow time.

If steering is ongoing and a movement from 5 GHz to 2.4 GHz band occurs, probe responses are allowed for this client until the steering is complete. Also, if the client is already associated in 2.4 GHz band, probe responses are allowed for this client.

With the probe response suppression feature configured and with client RSSI below configured RSSI threshold (the ‘Delay24GProbeRSSIThreshold’ parameter is set in lbd configuration file to a value considerably lower than current client’s RSSI on AP), probe responses are sent for the client. With the probe response suppression feature configured and with client RSSI above configured RSSI threshold (the ‘Delay24GProbeRSSIThreshold’ parameter is set in lbd configuration file to a value considerably higher than current client’s RSSI on AP), probe responses are not sent for the client for Delay24GProbeMinReqCount times in Delay24GProbeTimeWindow time.

## 16.5 SSID steering

This section describes the SSID steering algorithm implemented on QSDK platform. Due to the lack of configuration of SSID preference on the client devices, the devices may connect to a less preferable SSID where a more preferable SSID is available.

A station might not achieve sufficient bandwidth if it connects to public Wi-Fi in the home, despite its private Wi-Fi SSID being available. Thus, a mechanism needs to be established to ensure that devices that have been authorized on the private Wi-Fi will always connect to it rather than the public Wi-Fi SSID.

For this to happen, configure the virtual access point as either private or public. Based on the configuration of the VAP the association of the client is either allowed or denied.

### Algorithm input and output

Algorithm Inputs per VAP info:

Configuration of the VAP; whether it is configured as public or private.

Algorithm output:

Access policy per client; based on the parameters (public/private) given as input.

### Problem description

VAPs are classified as private or public and when a client tries to associate to the private VAP then it is allowed to associate to it. After that if the same client tries to associate to a public VAP then the association is denied.

It means that any client which has ever been connected to private VAP at least once will be denied association on the public VAP.

Configuration will be erased on reboot and AP is again sent into self-learning mode to learn private and public clients.

### Design principle

Consider two groups - one each for private Wi-Fi and public Wi-Fi and assume that the VAPs are configured as either public or private.

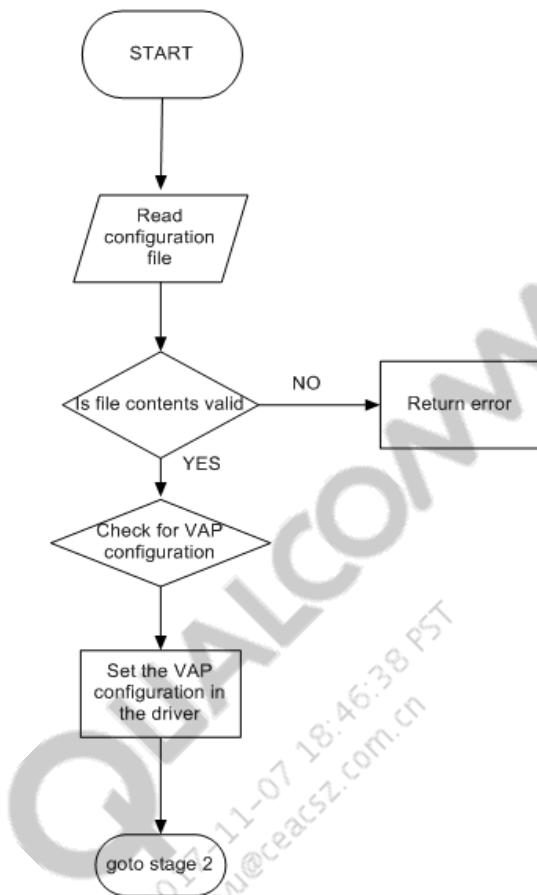
There is a Daemon running, that listens to the socket event sent by driver to user space

Depending on the VAPs configuration if there is an association request for a private VAP from a station then an association event is sent to the Daemon. Upon reception of this event in Daemon, the MAC address of that particular station is put in the ACL list of all the public VAPs on the same AP. Hence, if the station tries to associate to any public VAP on the same AP subsequently, then the association of that station is denied on the public VAPs of that AP.

The station will not be allowed to connect to the public VAP on that AP if it has been connected to the private VAP, at least once.

### Implementation

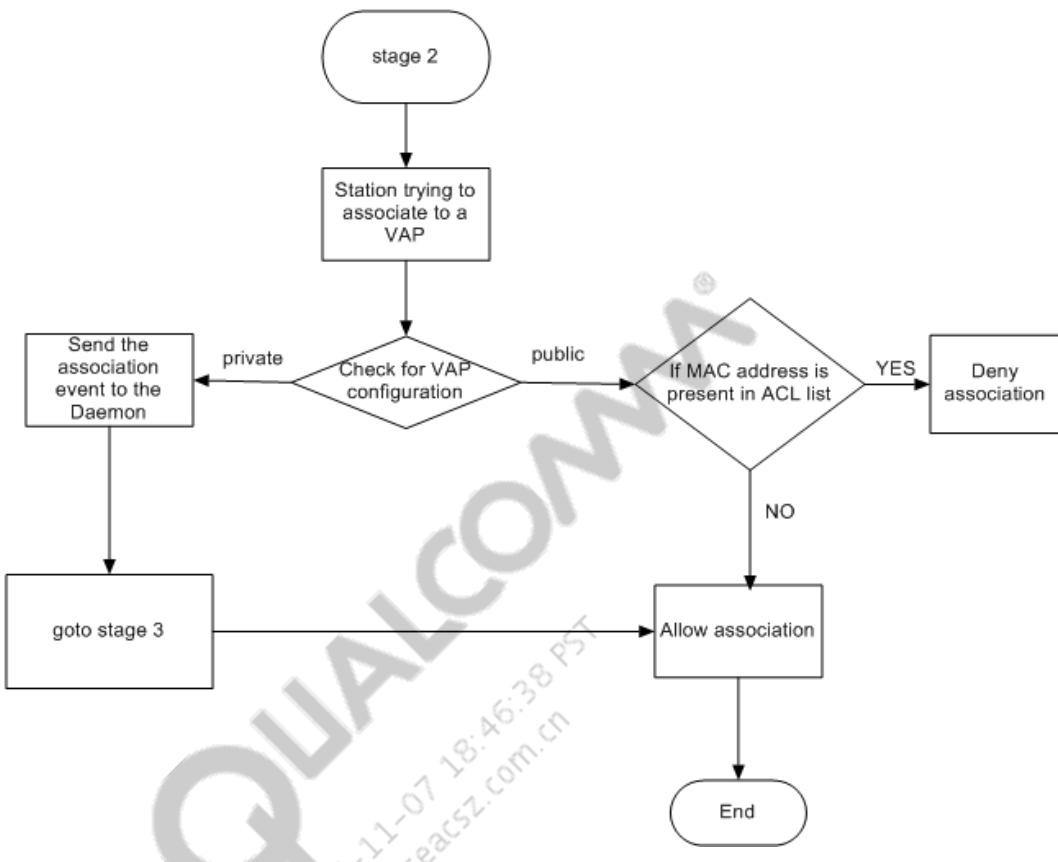
First, the daemon reads the content in the configuration file and checks for valid configuration and then sets the VAPs configuration according to the content.



**Figure 16-4 Implementation - Step 1**

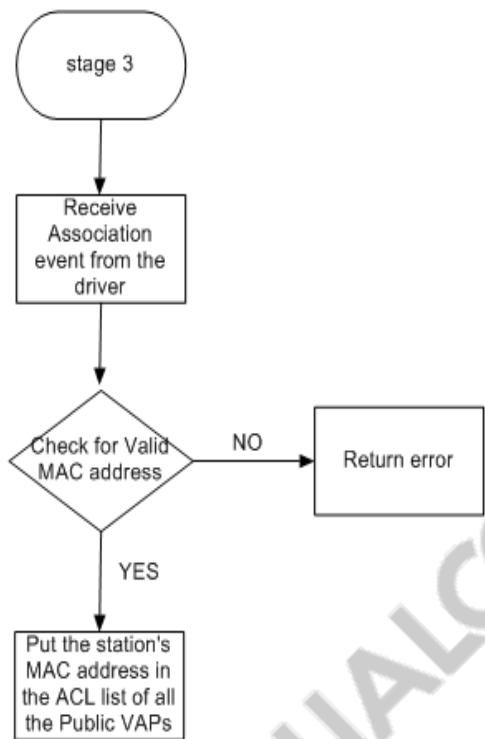
After that, when a station tries to associate to a VAP and if it is a private VAP then an event is sent from the driver to the Daemon and then that station's MAC address is put in the ACL list of all the public VAPs in that AP.

If the station tries to connect to a public VAP then the MAC address is checked in the ACL list and if it is present then the association is denied. If the MAC address is not present then association is allowed, as shown in [Figure 16-5](#).



**Figure 16-5 implementation - Step 2**

An association event is sent to the daemon from the driver whenever a private VAP tries to associate to it. And when this event is received the MAC address is checked whether its valid and it is put in the ACL list of all the public VAPS in that AP, as shown in [Figure 16-6](#).



**Figure 16-6 Implementation - Step 3**

## 16.6 Disable Wi-Fi SON blacklist functionality in band steering

The blacklist functionality from band steering is made available as a configurable option. To enable or disable this functionality, a configuration parameter called AUTH\_ALLOW is added to the lbd config file. When this feature is enabled, a new IOCTL, "WLANIBSTEERCONTROL\_AUTH\_ALLOW", is sent to the driver which will set a new flag across the MAC address present in the ACL list in the driver.

This IOCTL is sent along with the other IOCTL that sets the probe response for withholding flag. The IOCTL to clear the Auth Allow flag is also sent when the probe response withholding flag is cleared.

When the event ATH\_EVENT\_BSTEERING\_DBG\_TX\_AUTH\_ALLOW is received from the driver then the print indicating that the Authentication is allowed due to the setting of this flag.

In the driver, a new flag "IEEE80211\_ACL\_FLAG\_AUTH\_ALLOW" is added in the ACL entry. When this flag is set, Authentication response is allowed. This flag is set or cleared whenever the lbd sends an ioctl to the driver for the particular MAC address.

A wifitool command is also present to set/clear the IEEE80211\_ACL\_FLAG\_AUTH\_ALLOW flag for the MAC address present in the ACL list. When there is Authentication request for the particular MAC for which the AUTH ALLOW flag is set then Authentication is allowed.

When authentication is allowed due to the setting of the flag then an event ATH\_EVENT\_BSTEERING\_DBG\_TX\_AUTH\_ALLOW is sent to the lbd which will print the corresponding message.

The configuration parameter to enable the blacklist functionality in band steering is present in the config file. Under the IdleSteer configuration section of the config file, set the AUTH\_ALLOW option as 1 (enable) or 0 (disable).

Enter the UCI command as follows:

```
uci set lbd.IdleSteer.AUTH_ALLOW=1 uci commit lbd
/etc/init.d/lbd start
```

Enabling this feature might bring down the steering success rate. Steering might fail for those clients that would send an AUTH directly on previously serving band when being steered. The steering success rate characterization and tuning would be ultimately left to the customer.

When this feature is enabled using the configuration parameter, then it means it is applicable for both the bands. In other words, this is a global feature that is applicable across all Radios & cannot be enabled/disabled on individual radios.

The T\_Steering timer logic is not affected. The timer is started whenever the First Authentication rejection happens. This feature can be enabled or disabled only through lbd. This feature cannot be configured using command-line utilities such as iwpriv or wifitool. The lbd daemon is restarted, whenever this configuration parameter is changed so that the latest changes becomes effective.

## 16.7 Band steering in QWRAP mode

Starting with QCA\_Networking\_2017.SPF.5.0, band steering is supported in Qualcomm wireless repeater access point (QWRAP) mode. The band steering feature is implemented in a daemon named the Load Balancing Daemon (lbd). In QSDK-based platforms, the /etc/init.d/lbd script is responsible for taking the corresponding UCI configuration (found in /etc/config/lbd) and generating the format that can be understood by the daemon itself. It also determines the set of Wi-Fi interfaces that must be managed by lbd.

When WLAN interfaces are created from whc-interface.sh script and when AP operates in QWRAP mode, the script is modified to add AP VAP interface name into the WLAN interfaces. A ping loss of approximately 2~3 seconds is observed when band steering is successful. This behavior occurs because the proxy STA needs to reconnect whenever STA is moved from 2.4G to 5G or moved from 5G to 2.4G band.

For clients that support 802.11k/v, band steering is seamless and association priority is for 5 GHz band. For legacy clients, idle band steering is supported.

Keep the following points in mind while configuring band steering with QWRAP:

- Only band steering and not AP steering is supported.
- User must start lbd using UCI parameters and repacd support is not available.
- With added functionality in QWRAP module, the MAC address of a node does not change even when the node is steered from one band to another band. This mechanism enables the

flows to be maintained, even when steering happens and less number of ping timeouts are observed in such scenarios.

### 16.7.1 Verify single AP band steering daemon is running, stadb logs and bandmon logs

This procedure verifies that APUT successfully starts daemon and shows stadb and bandmon logs properly. Consider a scenario in which 2.4GHz channel is configured as 6 or 11 to verify whether band utilization is being updated properly. Configure channel 156 on 5G to verify whether lbd daemon is starting. AP is configured in DBDC mode, WPA2-PSK security (same SSID and key on both the bands) is used, and wireless client must be dual band clients.

1. Start the APUT with both bands enabled (QWRAP DBDC mode).  
The configuration is successful and can ping to Root AP.
2. Enable lbd daemon on the APUT.
3. Verify whether the band steering (lbd) daemon is running using the command "ps | grep lbd".  
The lbd daemon is running.
4. Open a Telnet session to the APUT on port no.7787 for the Band Steering daemon debug logs and enable dump logs.

**NOTE** Open a Telnet session to the APUT on port no.7777 for the -Hy-Fi daemon (hyd) debug logs.

5. Connect two wireless clients, say STA1 and STA2, to 5G and ping packets sent to check the connectivity between them. STA1 and STA2 connect successfully
6. In the Telnet session, check the "stadb s" command output, which displays the "In Network" field as yes for STA1 and STA2 MAC addresses.  
The "stadb s" command output displays Bands=25 and ChanID=<current 5G channel> under Assoc.
7. Inject traffic from console to the clients and see band utilization using 'bandmon s'.  
The command displays the values greater than 0 for client connected channel.

### 16.7.2 Verify preassociation steering 11k/v clients based on 2.4GHz overload detection

This procedure verifies pre-association band steering based on overload detection from 2.4 GHz to 5 GHz with 11k/v supported clients. AP is configured in DBDC mode, WPA2-PSK security (same SSID and key on both the bands) is used, and wireless client must be dual band clients.

1. Start the APUT with both bands enabled (QWRAP DBDC mode).  
The configuration is successful and can ping to Root AP.
2. Enable lbd daemon on the APUT.
3. Verify whether the band steering (lbd) daemon is running using the command "ps | grep lbd".  
The lbd daemon is running.

4. Open a Telnet session to the APUT on port no.7787 for the Band Steering daemon debug logs and enable dump logs.
5. Connect two wireless clients say STA1 (mono-band support, non-11k/v) STA2 (dual-band, 11k/v supported) to the 2.4GHz band and run ping to check the connectivity between them. STA1 and STA2 successfully connected to 2.4 GHz band of APUT.
6. Disconnect STA2. STA2 is disconnected.
7. In the Telnet session, check the “stadb s” command to see the entries for both the clients. The “stadb s” command output shows STA1 as connected and STA2 as disconnected.
8. Inject traffic onto 2.4GHz using iperf/chariot running between STA1 and the AP so that 2.4 GHz band is overloaded. Traffic from AP to STA1 runs successfully
9. Verify AP detects that the 2.4GHz is overloaded (from debug log msgs from the AP). The “bandmon s” command output in the telnet session confirms 2.4 GHz band is overloaded.
10. Move the test wireless client STA2 close to the AP and start scanning in client side, so that 5 GHz signal is strong, ensure 5GHz RSSI > (OffloadingMinRSSI i.e. 20 dB). The “stadb s bss” command output confirms 5GHz for STA2 is greater than 20 dB
11. Connect STA2 to the APUT. In the sniffer trace verify that when client sends Authentication frame on 2.4GHz, AP rejects it, and that the client successfully associates with 5GHz.
12. Also confirm the pre-association band steering from telnet session debug logs. Check ‘steerexec s’ for STA2 entry.

## 16.8 Configure band steering per SSID pair

To ensure bandwidth fairness or guaranteed throughput, when a client is being steered from one VAP to another VAP on the same ESS, all VAPs on other ESSes must be blacklisted. A mechanism to disable band steering on specific SSIDs is implemented. The feature is disabled by default and will be enabled using a config option in the load balancing daemon (lbd) config file. This functionality operates with multi-band steering and not with multi-AP steering. The user specifies the SSIDs to be part of band steering. All other SSIDs are excluded from band steering and do not participate in band steering.

'BlacklistOtherESS' is the option added in the lbd config file to enable/disable this feature. The default value is `BlacklistOtherESS '0'`, which signifies that the band steering capability per SSID pair is disabled. The `BlacklistOtherESS '1'` value signifies that the feature is enabled.

With this feature enabled in the config section of the lbd config file, blacklist entries are installed on all VAPs, except for the target VAP (candidate VAP) during band steering.

### Preassociation band steering

With this feature disabled or not supported, pre-association steering matches the channel number and installs blacklist entries on all VAPs on the same band. With this feature enabled, the pre-association steering installs blacklists on other extended service set (ESS) VAPs, except for the target VAP. To install blacklists on target VAP, the target ESS information must be sent down to the wlanif while the channel performs the blacklisting. The stadb module stores the last serving ESS,

which must be passed on to the steerexec module. Later, the steerexec module sends down the information the wlanif module so that it can do the right blacklisting.

### Post-association band steering

With this feature disabled or not supported, post-association steering matches the ESSID and install blacklists only on the specific VAP that matches the ESSID. With this feature enabled, post-association steering installs blacklists on all VAPs except for the candidate VAP. The target ESS information is already available in this function. Therefore, the function arguments are not changed.

User can specify an option to enter the SSIDs that must be included in band steering. This list of SSIDs must be entered in the lbd config file. All other SSIDs except the list of SSIDs mentioned in the config file are excluded from band steering algorithm.

By default, the MatchingSSID list is empty, which indicates that all the interfaces are to be included. An empty MatchingSSID list and a list that contains matching SSID values cannot coexist. Therefore, to configure interfaces that match a particular SSID, issue the uci del\_list lbd.config.MatchingSSID=" command (for example, uci add\_list lbd.config.MatchingSSID='ssid\_name').

### Sample configuration scenarios

If a user brings up the AP in MBSSID mode (2 VAPs on each radio), enables the feature, and pre-association steering is triggered, blacklist rules are installed on all VAPs except for the target/candidate VAP. Similarly, if a user brings up the AP in MBSSID mode (2 VAPs on each radio), enables the feature, and post-association steering is triggered, the blacklist rules are installed on all VAPs except for the target or candidate VAP (as displayed with the ACL filter using iwpriv command).

If a user brings up the AP in MBSSID mode (3-4 VAPs on each radio), enters the SSID names that must be included in band steering, and connects a client to an SSID that is excluded from band steering, then this client is not steered by the band steering daemon.

The excluded VAPs are present in a separate list and the blacklisting is installed/cleared during preassociation or post-association steering on all the excluded VAPs.

## 16.9 AP steering for legacy clients

This section describes the AP steering for legacy clients by use of RSSI Monitoring using Smart Monitor mode. Multi AP with one 2.4 GHz and/or one or two 5 GHz radio(s). The radios operate concurrently on different channels. This section describes the design rationale and the actual steering. The purpose of this steering is to position the clients with improved WiFi Network performance by steering the clients to a better-serving AP through a better RSSI. A client for steering will be a legacy client, a client that does not support 802.11k beacon measurement, a client that is certified by the upcoming Wi-Fi Alliance MultiBand Operation (MBO) certification, or a client that fails for consecutive attempts of 802.11k beacon report.

Support is implemented for APs to steer legacy clients, which do not support 802.11k beacon measurement and MBO compliance. These legacy clients are clients that support 802.11v basic

service set (BSS) transition management (BTM) and do not support Radio Resource Management (802.11k). Only idle steering and not active steering is supported.

This section assumes the scenario where there are multiple APs (base AP and range extenders – Wi-Fi routers at home) operating as single-band/dual-band/dual-concurrent/multi-concurrent AP, serving a number of single-band and dual-band single radio clients. The assumption is that the AP has one 2.4 GHz radio and one or more 5 GHz radios.

A typical client chooses its serving AP based on beacon signal strength. When the client roams away from the serving AP, its signal strength weakens and APs might exist (base AP or range extenders) with better signal strengths. The clients may choose or stay on a AP even when better RSSI is available on nearby AP. There is a high chance that the clients do not steer to better RSSI APs. This creates non-optimal system performance across client and AP.

- A client at the coverage edge using MCS0/1, which takes up most of the medium and degrades other clients that happen to be on the same band.
- The clients may select or stay with a AP even when they receive better service/data rate on another AP.
- Clients with suboptimal signal strength with AP degrades voice/video performance.

The preceding cases create inefficient utilization of the available bandwidth that leads to poor performance of the system and degraded user experience.

Because each client puts a different load on the system based on its RF condition and data usage, they are not all equal. Clients do not easily steer among APs. Instead, AP must monitor the actual strength of the client based on the RSSI to make a better steering decision and request the client to steer.

AP must steer legacy clients (that do not support 802.11k beacon measurement or that fail for consecutive attempts of 11k beacon report). However, a limitation exists in dealing with sticky legacy clients.

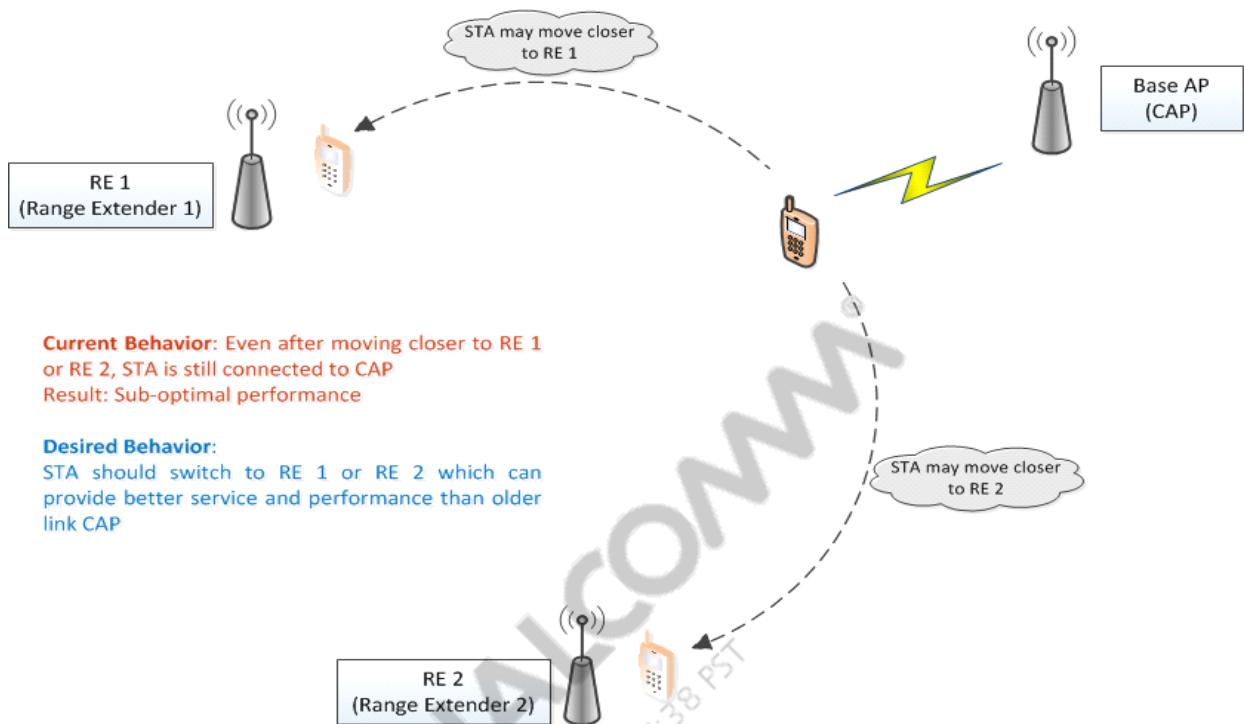
### 16.9.1 Design principles

The AP steering strategy is determined by the RSSI of the client via Smart Monitor mode trigger and use it for steering clients that are stuck with sub-optimal AP. The AP attempts to steer a client to provide a higher data rate for the client.

Client connects to APs (REs/CAP) and then moves closer to other APs (REs / Base). Even after moving closer to Base, some clients hang on to their old AP connection and do not try to connect to better Base AP option, thereby degrading throughput performance.

Allowing serving-AP to trigger RSSI monitoring of the client (MAC Address) by all other APs in the network, to find if this client is better served on some other AP. Based on the results, serving AP retains or steers the client to better serving AP.

Client serving AP with less RSSI must steer to better AP in the network. Serving AP must collect RSSI stats of the client from all APs in the network and steer the client.



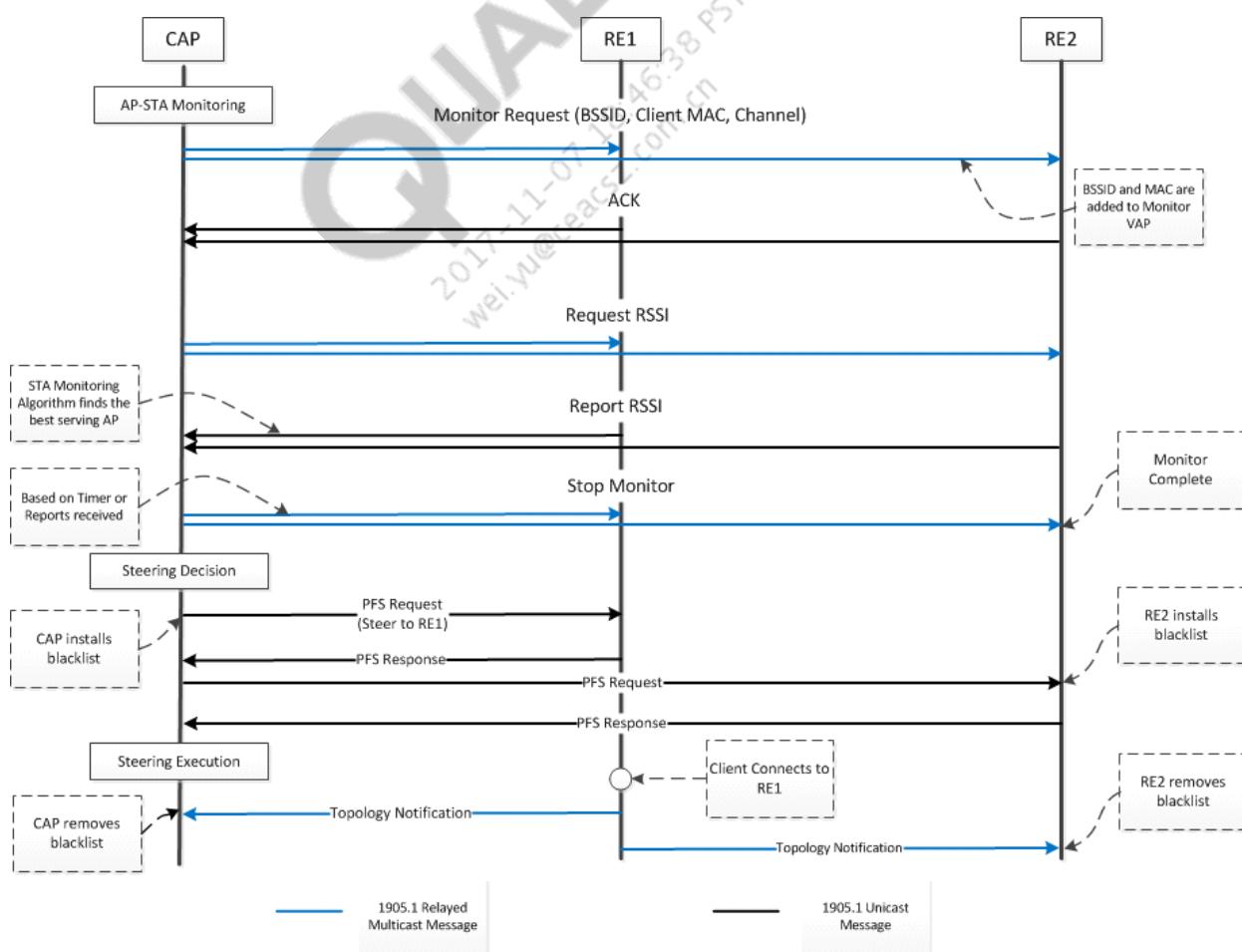
Each node of an AP in the network contains a Smart Monitor VAP on both radios by default (2G and 5G). Smart Monitor VAP, by default, will have no <MAC addr, BSSID> filter set (that is, no packets received on this VAP). Serving AP will have a Monitor RSSI Threshold. When RSSI for a connected STA falls below this threshold, serving AP will trigger Monitor RSSI event. Monitor RSSI will be passed to all other APs in the network. APs receiving Monitor RSSI event will add <STA Mac, BSSID> filter to Smart Monitor VAP and start receiving up-streamed packets on air by this STA MAC for given BSSID. These packets will help determine RSSI between monitoring node and monitored STA. Monitoring nodes periodically send RSSI info to serving AP. Serving AP will decide if this connected STA can be better served by any other AP. If better, start steering process (which also ends the monitoring process on all nodes as steering event is heard by all). If not, serving AP may end monitoring after certain time period by sending Stop Monitor Event.

### 16.9.1.1 Steering criteria

Serving-AP (node initiating Monitor RSSI Request) contains a timer control, within which it needs to make a steering decision. Until this timer expires, the serving-AP continues to receive Monitor RSSI Reports from other APs. Either before expiry or upon expiry of this timer, serving AP can make STA steering decision based on received RSSIs. Serving AP gets current STA's RSSI using the load balancing daemon (lbd) (QoS Null-ACK method). Monitor mode converts RSSI in same format as the format that the lbd maintains. If no decision is reached upon expiry of the timer or other RSSIs are not better for steering, serving-AP sends Stop Monitor Event to all nodes to stop RSSI monitoring on other nodes.

### 16.9.1.2 Steering algorithm

1. A Monitor mode VAP is created across all nodes for APs in the network.
2. The serving AP monitors the RSSI of the associated client.
3. After the RSSI falls behind the threshold, serving AP initiates a Start Monitor timer.
4. Serving AP also initiates monitoring for clients which had failed to report 11k beacon report for certain consecutive attempts.
5. Non-serving APs in the network add the serving AP BSSID and STA MAC to Smart Monitor VAPs to record RSSI.
6. The serving AP maintains the list of clients for which non serving AP failed to monitor.
7. The serving AP requests RSSI stats for the associated client from other APs in the network by 1905 message.
8. Non-serving APs reports RSSI to the serving AP through 1905 message.
9. The serving AP collects RSSI and makes steering decision through RSSI threshold difference.
10. The serving AP sends cancel Monitor event and triggers steering event.



### 16.9.1.3 Functionality

- MonitorRSSIThreshold—Serving AP monitors the RSSI threshold of the associated clients on a periodic basis.
- MonitorMax11kFailure—Serving AP monitors maximum consecutive 11k beacon report failures and requests for monitoring of those clients.
- MonitorRequest—Serving AP starts timer and requests event to all other APs in the network with Client MAC and BSSID of serving AP. Serving AP monitors the timer and awaits RSSI report.
- MonitorResponse—Non-serving AP starts monitoring, starts timer and initiates host driver ioctls/calls for monitoring. Non-serving AP responds with an acknowledgment to serving AP.
- MonitorRSSIRequest—Serving AP requests for RSSI information to all other APs in the network.
- MonitorRSSIReport—Non-serving AP reports RSSI information to serving AP.
- StopMonitor—Cancel Monitoring timer, initiate stop monitor event to other APs in the network, Find best serving AP, Prepare steering decision and remove RSSI storage information.

### 16.9.2 Configuration

After the image is loaded into the flash memory and the device is configured (including Smart Monitor Configuration as listed later in this subsection as an example), reboot AP.

Edit /etc/config/hyd for the following and restart hyd.

The following is the default configuration:

```
config Monitor 'Monitor'
    option DisableMonitoringLegacyClients '1'
    option DisableSteeringInActiveLegacyClients '1'
    option DisableSteeringActiveLegacyClients '1'
    option MonitorTimer '60'
    option MonitorResponseTimeout '5'
```

1. Configure *DisableMonitoringLegacyClients* as 0 for smart monitor to be functional.
2. Configure *MonitorTimer* as 60 for periodic monitoring.
3. Configure *MonitorResponseTimeout* as 5 for better monitoring of multiple clients.
4. Configure *DisableSteeringInActiveLegacyClients* as 0 for monitoring/steering of Inactive Legacy clients.
5. Configure *DisableSteeringActiveLegacyClients* as 0 for monitoring/steering of Active Legacy clients.
6. Enter the \$ /etc/init.d/hyd restart command to restart hyd.

**NOTE** For this feature to work, it is necessary to create monitor VAPs on the two radios used for client connection, which are wifi0 and wifi1 in this case. If these VAPs are

different on the board in a particular network, the user must perform appropriate changes.

### Smart Monitor configuration

```

uci set wireless.wifi0.repacd_auto_create_vaps=0
uci set wireless.wifil.repacd_auto_create_vaps=0
uci add wireless wifi-iface
uci set wireless.@wifi-iface[4].device=wifi0
uci set wireless.@wifi-iface[4].network=lan
uci set wireless.@wifi-iface[4].mode=ap_smart_monitor
uci set wireless.@wifi-iface[4].ssid=ap_smart_monitor1
uci set wireless.@wifi-iface[4].encryption=none
uci set wireless.@wifi-iface[4].neighbourfilter=1
uci set wireless.@wifi-iface[4].set_monrxfilter=1
uci set wireless.@wifi-iface[4].disable=0
uci set wireless.@wifi-iface[4].wsplcd_unmanaged=1
uci set wireless.@wifi-iface[4].repacd_security_unmanaged=1

uci add wireless wifi-iface
uci set wireless.@wifi-iface[5].device=wifil
uci set wireless.@wifi-iface[5].network=lan
uci set wireless.@wifi-iface[5].mode=ap_smart_monitor
uci set wireless.@wifi-iface[5].ssid=ap_smart_monitor2
uci set wireless.@wifi-iface[5].encryption=none
uci set wireless.@wifi-iface[5].neighbourfilter=1
uci set wireless.@wifi-iface[5].set_monrxfilter=1
uci set wireless.@wifi-iface[5].disable=0
uci set wireless.@wifi-iface[5].wsplcd_unmanaged=1
uci set wireless.@wifi-iface[5].repacd_security_unmanaged=1
uci commit
reboot -f

```

### Steering for 802.11k unfriendly clients

Edit /etc/config/lbd for the following and restart hyd. The default configuration is as follows:

```

config Estimator_Adv 'Estimator_Adv'
    option Max11kUnfriendly '10'

```

*Max11kUnfriendly* must be configured to ‘n’ for clients to be initiated with Smart Monitoring if 11k beacon report fails ‘n’ consecutive attempts.

### 16.9.3 Enhanced AP steering of 802.11k unfriendly clients as legacy clients

Starting with QCA\_Networking\_2017.SP5.0.3 CS, the capability of AP steering for legacy clients is extended to clients that fail for consecutive attempts of 802.11k beacon reports. For such 11k-unfriendly clients, edit /etc/config/lbd for the Max11kUnfriendly option under the Estimator\_Adv 'Estimator\_Adv' config section and restart hyd. The Max11kUnfriendly option must be configured to ‘n’ for clients to get initiated with Smart Monitoring if 11k beacon report fails ‘n’ consecutive attempts. The default value of this option is 10.

|               |               |                  |   |    |
|---------------|---------------|------------------|---|----|
| Estimator_Adv | Estimator_Adv | Max11kUnfriendly | Enable the capability of AP steering for legacy clients to be extended to clients that fail for consecutive attempts of 802.11k beacon reports. For such 11k-unfriendly clients, edit /etc/config/lbd for the Max11kUnfriendly option under the Estimator_Adv 'Estimator_Adv' config section and restart hyd. The Max11kUnfriendly option must be configured to 'n' for clients to get initiated with Smart Monitoring if 11k beacon report fails 'n' consecutive attempts. | 10 |
|---------------|---------------|------------------|---|----|

#### 16.9.4 AP steering for legacy clients on QCA9886 chipsets

Starting with QCA\_Networking\_2017.SPF.5.0.3, AP steering for legacy clients using RSSI monitoring in Smart Monitor mode is supported on QCA9886 chipsets and on any platform that uses the QCA9886 radio (for example, IPQ4019 (ARM processor) + QCA9886). The purpose of this steering is to position the clients with improved Wi-Fi network performance by steering the clients to a better-serving AP through a better RSSI. A legacy client is a client that does not support 802.11k beacon measurement or a client that is certified by the upcoming Wi-Fi Alliance MultiBand Operation (MBO) certification.

### 16.10 Display the hyd and lbd versions

The Hy-Fi daemon (hyd) and load balancing daemon (lbd) version information is essential to identify the images of these processes or daemons that are used in the release of a software product. While using different software images, it is difficult to determine the version of the hyd that is being used in a particular software image release. The mechanism to display the hyd and lbd versions is implemented. The hyd and lbd versions are displayed in the output of explicit commands introduced for this purpose.

The hyd version is determined by user space command with the -v argument. The command retrieves and displays the version of the hyd and lbd. The lbd version is displayed only if the WLB module is enabled as part of hyd. The lbd version command retrieves only the version of the lbd module that is used. Enter the hyd -v command to view the hyd and lbd versions. Enter the lbd -v command to view the lbd version.

The following is a sample output of the hyd -v command:

```
root@OpenWrt:/# hyd -v
hyd v1.0
User space daemon for HYD
Copyright (c) 2011 Qualcomm Atheros, Inc.
All Rights Reserved.
Qualcomm Atheros Confidential and Proprietary.

lbd v1.0
User space daemon for LBD
Copyright (c) 2011 Qualcomm Atheros, Inc.
All Rights Reserved.
Qualcomm Atheros Confidential and Proprietary.
```

The following is a sample output of the lbd –v command:

```
root@OpenWrt:/# lbd -v
lbd v1.0
User space daemon for LBD
Copyright (c) 2011 Qualcomm Atheros, Inc.
All Rights Reserved.
Qualcomm Atheros Confidential and Proprietary.
```

## 16.11 Soft-blocking for safe connections

This section describes the soft-blocking functionality. Similar to the SSID steering mechanism, the goal of this capability is to force private users to connect to private SSID, instead of a public SSID. The soft-blocking capability is a safety mechanism; it tries to allow the client to connect, if the public SSID is continued to be connected in a period.

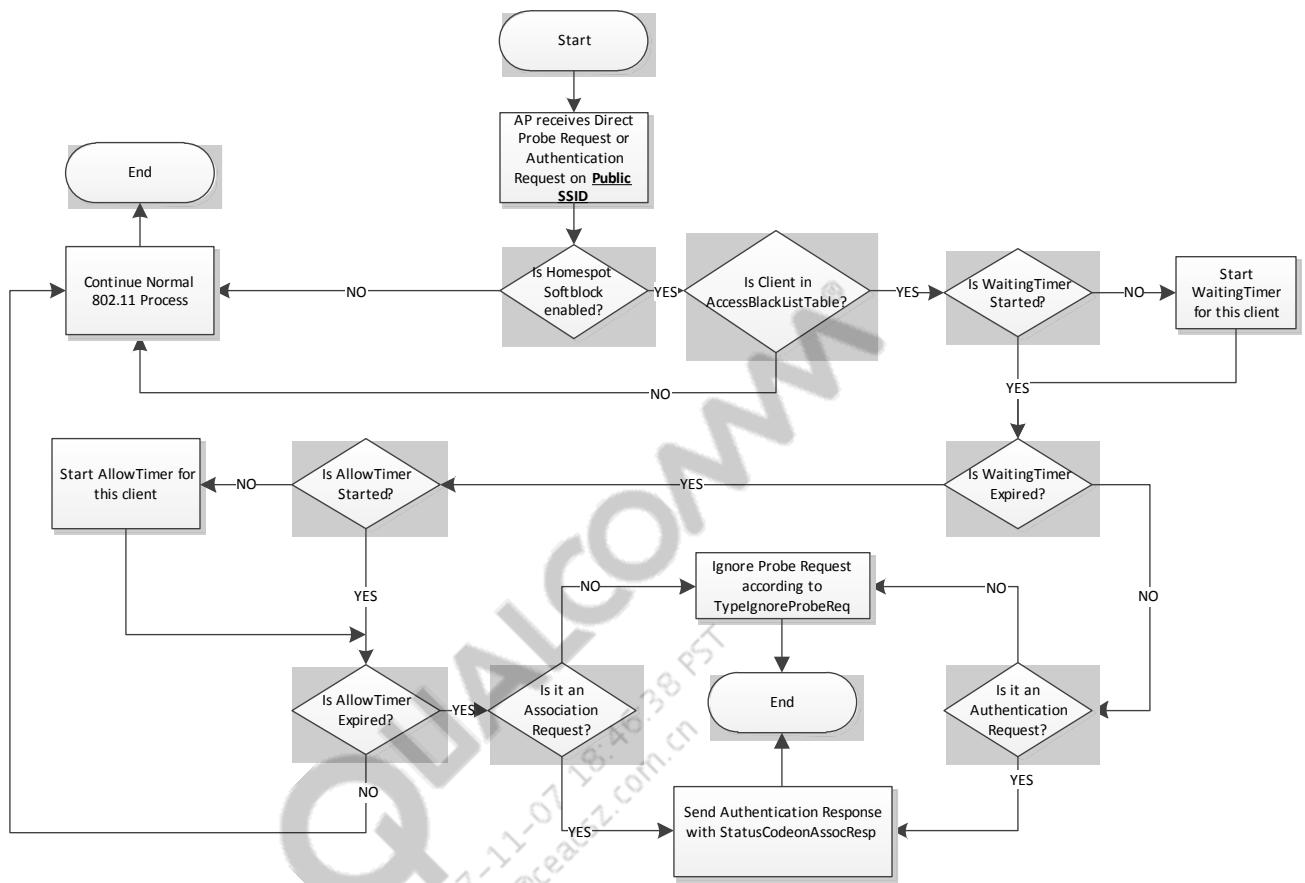
The following sequence of events occur with the soft-blocking functionality:

1. Client associates to private SSID
2. AP will add client to the blacklist for Open/EAP SSID
3. If client associates on Open/EAP Public SSID, check blacklist for client's MAC address. If MAC address is on the blacklist, perform a 'Soft Block' to deny the client to authenticate with Public SSID.
4. MAC address will remain in blacklist of Open/EAP SSID until it is manually cleared.

Association with the private SSID must reset the SoftBlockAllowTimer and SoftBlockWaitingTime to enable a client to connect to private SSID; the client does not abruptly jump back to the public SSID.

When the SoftBlockAllowTimer has expired, both the SoftBlockWaitingTimer and the SoftBlockAllowTimer must be reset to the state of "NOT STARTED" so that the whole SoftBlock process is restarted when the client retries to connect to the Public SSID.

The following flowchart illustrates the soft-blocking process:



When a Wi-Fi client sends a Probe Request or an Authentication Request to associate (or re-associate) with the Public SSID, then the following operations occur:

1. A check is performed to determine whether the Wi-Fi client is in the Public SSID Blacklist table.
2. If client is listed, start a timer for Time\_Period as defined as “cmdot11SoftBlockWaitingTime” (default=10 secs).
3. HGW must not send any Probe Responses or send Authentication Responses with a specific set Status Code (default = 1- “unknown reason”) of the Public SSID until “cmdot11SoftBlockWaitingTime” period has elapsed.
4. If Wi-Fi client continues to attempt to associate with Public SSID, HGW should send a normal Probe Response or Authentication Response to the client for Public SSID and grant WiFi access as normal for a limited period as defined as “cmdot11SoftBlockAllowTimer”.

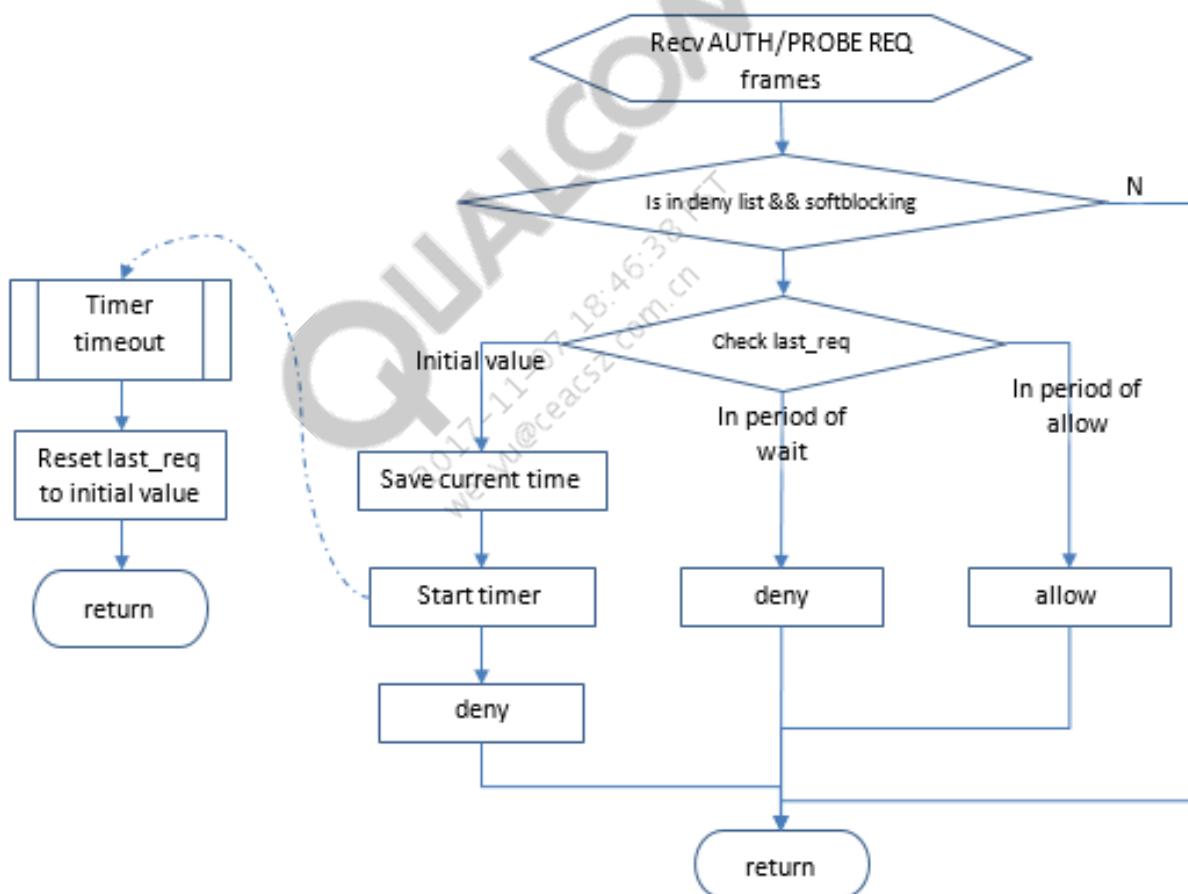
### 16.11.1 Design guidelines

There are two timers in the soft-blocking process, SoftBlockWaiting and SoftBlockAllow. This is the main difference from the SSID steering methodology. With the two timers, it offers a protected and secure option to allow a client continually connecting the public SSID for some reason. The DENY list of the ACL function is used. When authentication and probe requests are sent, if the

MAC address is in the deny list and with soft-blocking flag, the timers are set up to control the allow and blocking.

There is an ACL for each VAP. Add a new ACL\_FLAG\_SOFTBLOCKING to indicate that the ACL entry is a SOFTBLOCKING entry. Then, when receiving AUTH or direct PROBE REQ frames, check the mac address in DENY list and check the flag. If found, check the last request timestamp of the entry. If the last request is the INITIAL VALUE, save the current timestamp and start a timer; if last request is in the period of WAIT time, block it; if in the period of ALLOW time, allow it.

Two lists are available; because the LIST\_1 might be used as ACL and hostapd might fetch the entries to block it, LIST\_2 is used as the softblocking list.



### 16.11.2 Commands

- Enter the `iwpriv athX addmac_sec <mac>` command to add specified MAC addresses to the secondary access control list (ACL).

- Enter the iwpriv athX maccmd\_sec 2 command to instruct how the ACL is used to limit access the AP.
- Enter the iwpriv athX wait\_time <0-100000> and iwpriv athx g\_wait\_time commands to set/get the parameter of the wait time in microsecond, default is 10000.
- Enter the iwpriv athX allow\_time <0-100000> and iwpriv athx g\_allow\_time commands to set/get the parameter of the allow time in microsecond, default is 15000.
- Enter the wifitool athX softblocking <dstmac> <0/1> and wifitool athX softblocking\_get <dstmac> commands to set/get the enable flag of the dstmac. Setting the flag resets the timer to the initial value.

**Table 16-11 Softblocking parameters**

| Parameter                                | Command   | Description   |
|--|---|---|
| wait_time <0-100000><br>athx g_wait_time | iwpriv athX wait_time <0-100000><br>iwpriv athx g_wait_time   | Set/get the parameter of the wait time in microsecond, default is 10000.  |
| allow_time <0-100000><br>g_allow_time    | iwpriv athX allow_time <0-100000><br>iwpriv athx g_allow_time | Set/get the parameter of the allow time in microsecond, default is 15000. |

**Table 16-12 Softblocking parameters**

| Parameter  | Command  | Description   |
|--|--|---|
| softblocking <dstmac> <0/1><br>softblocking_get <dstmac> | wifitool athX softblocking <dstmac> <0/1><br>wifitool athX softblocking_get <dstmac> | <p>Set/get the enable flag of the destination MAC address (dstmac). Setting the flag will reset the timer to initial value.</p> <p>Similar to the SSID steering mechanism, the soft-blocking capability is to force private users to connect to private SSID, instead of a public SSID. The soft-blocking capability is a safety mechanism; it tries to allow the client to connect, if the public SSID is continued to be connected in a period.</p> <p>If client associates on Open/EAP Public SSID, check blacklist for client's MAC address. If MAC address is on the blacklist, perform a 'Soft Block' to deny the client to authenticate with Public SSID.</p> <p>MAC address will remain in blacklist of Open/EAP SSID until it is manually cleared.</p> |

# 17 Repeater Access Point Functions

This chapter describes the functions of an access point (AP) that works as a range extender (RE) or a repeater, such as Dual-Band Dual-Concurrent (DBDC) repeater, extender APs, Qualcomm Wireless Repeater Access Point (QWRAP), Power Amplifier PRe-Distortion (PAPRD), Wireless Distribution System (WDS), Proxy STA, and Wi-Fi Protected Setup (WPS) for range extenders and repeaters.

## 17.1 PAPRD

### 17.1.1 Theory of Operation

When a packet is transmitted over the over air, the signal can be distorted at various places. The transmit frame goes through the MAC, digital baseband, and finally a DAC (Digital-to-Analog Converter). This analog baseband signal is amplified before feeding to antenna. Ideally the Power Amplifier (PA) should have zero signal distortion. While designs with external PAs show very good linearity and low distortion, designs which use the internal PA show significant distortion at higher signal powers. This feature tries to address the distortion in the internal Power Amplifier of the transmitter. The PAPRD (Power Amplifier PRe-Distortion) algorithm is used to reduce this signal distortion digitally using a combination of software and hardware processes. The algorithm is applied only on the designs that use internal PAs. Example designs are HB112, HB116 and some AR934x/AR935x-based designs that use the internal PA of the radio.

Figure 17-1 shows an example of the distortion caused by the PA:

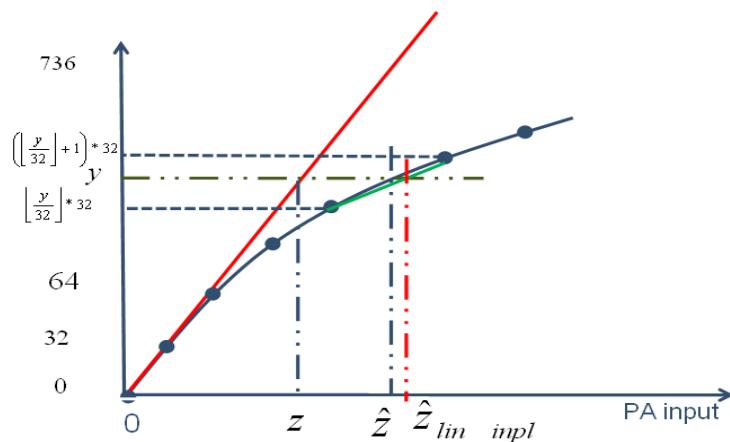
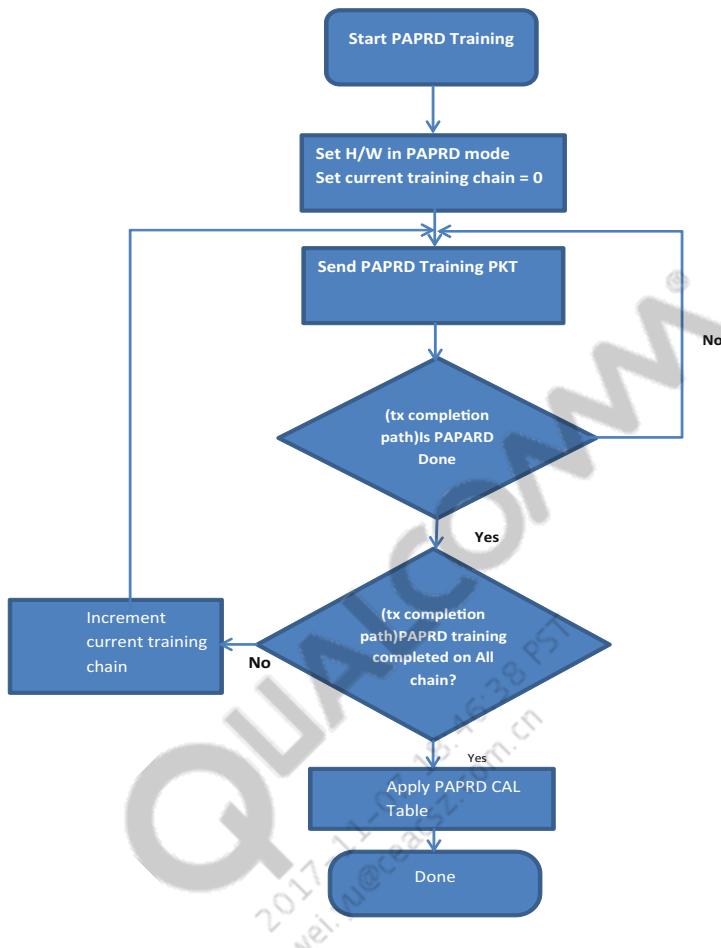


Figure 17-1 PA Distortion



**Figure 17-2 PAPRD Algorithm Flowchart**

### 17.1.2 Implementation

In general, the software needs to perform the following tasks to for the PAPRD algorithm:

- Set up chip in PAPRD training mode.
- Send training packets for one individual chain at a time.
- Capture hardware loopback signal samples.
- Run the software algorithm to derive the PAPRD Calibration Information.
- Store this PAPRD CAL information into software data structure.
- Apply captured PAPRD CAL to chip.
- Re-apply PAPRD CAL if chip is reset.

### 17.1.2.1 Enable PAPRD

The PAPRD feature can be enabled at compile time by defining the “ATH\_SUPPORT\_PAPRD” compile flag.

At runtime, the following conditions must be met for the PAPRD feature to be running:

- Registry entry ‘sc\_reg\_parm.paprdEnable’ must be true.
- HAL capability ‘HAL\_CAP\_PAPRD\_ENABLED’ must be true. This capability is stored in the EEPROM as part of the radio calibration step.

If either of aforementioned conditions is false, PAPRD is disabled.

### 17.1.2.2 PAPRD Training

The PAPRD training is closely tied to the generic calibration function, `ath_calibrate()`, which runs as a timer thread and starts various calibrations such as NF cal, IQ cal, and so on. This function runs every 2 seconds in the STA mode and every 200 ms in the AP mode until the calibration is done. After calibration is complete, `ath_calibrate` is called every 30 seconds in the STA and AP modes.

After successful completion of PAPRD training, the PAPRD per chain CAL table is saved and applied by calling ‘`ath_apply_paprd_table()`’. PAPRD CA L is retained for the current channel for which training was performed. If a channel change occurs, PAPRD training restarts on the new channel.

PAPRD training packets are sent by calling ‘`ath_sendpaprd_single()`’. The details of the training packet are as follows:

```

pkt len = 1800 bytes
Data rate: 54Mbps.
Frame Type: Data.
Number of PAPRD Training frame per chain: 1

```

### 17.1.2.3 Excessive PAPRD Failure

If the PAPRD training fails consecutively for more than 100 times for any reason, the PAPRD algorithm is disabled. During development and testing this issue has been seen due to faulty calibration, erased or uninitialized calibration of Wi-Fi component, incorrectly compiled code or loose antenna connection. In these cases the software disables this feature until next boot.

### 17.1.2.4 PAPRD Training Power

The Tx power limit ensures that power is within the FCC specification for a desired Tx rate and channel. For higher rates (such as MCS 23/15/14/7), Tx power is limited by the EVM requirement. However, for lower rates Tx power is limited by mask power.

For higher rates (MCS23/15/14/7) EVM should be less than -26. By enabling PAPRD, power can be increased for these rates while still achieving 2 or 3 dBm EVM improvement. That translates to a better range for higher rates.

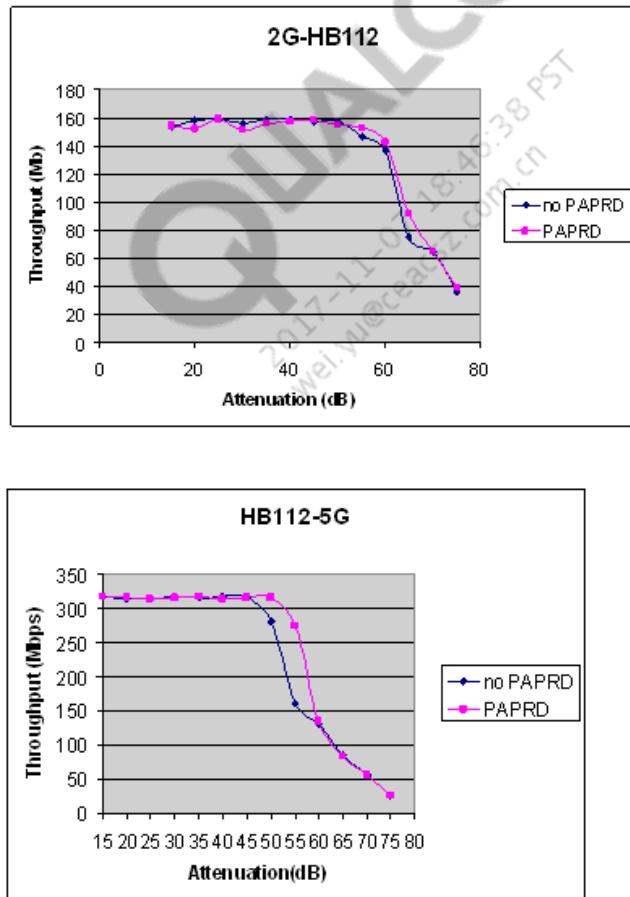
Currently PAPRD takes effect for following rates/spatial streams: MCS 23/15/14/7 for 3-chains designs such as HB112 and MCS15/14/7 for 2-chain designs such as HB116

To prevent regulatory violations in the target Tx power, which is higher for PAPRD rates than without PAPRD, the following additional steps are taken by software

- If PAPRD training is not complete, subtract 3 dBm from target power for PAPRD rates.
- After PAPRD calibration is done, update target power table with correct power number.
- If target power is changed by the upper layer (for example, vendors change target power using ‘ath\_hal\_settxpowlimit()’) and the delta is greater than 2.5 dBm between current power and default target power, then PAPRD training is skipped.

### 17.1.2.5 PAPRD Test Result

RvR test was run with and without PAPRD enabled and range improvement can be seen in [Figure 17-3](#).



**Figure 17-3 RvR Throughput with and without PAPRD**

### 17.1.2.6 Codebase

This section explains the various files that make up the PAPRD module. The files are split up by functionality to make the module easier to maintain.

- `ath_paprd.c`

This file is the main software interface to the rest of the WLAN driver. This contains – init/transmit/transmit completion/cleanup

- `ar9300_paprd.c`

This is the corresponding HAL file for PAPRD related functions. It contains code to put the chip into PAPRD training mode. It includes the capture PAPRD CAL table and writes PAPRD CAL table to BB registers.

### 17.1.2.7 Software APIs

- `ath_paprd_cal`

Set HAL into PAPRD CAL mode and send training packets.

- `ath_tx_paprd_init`

Checks PAPRD flags if enabled appropriately by HAL and registry and initialize PAPRD queue, PAPRD pkt and data structs.

- `ath_apply_paprd_table`

This call writes PAPRD CAL table to baseband registers.

- `ath_tx_paprd_complete`

This is called when a PAPRD packet is transmitted on air. This routine checks if PAPRD training has completed on this particular Tx chain and moved to next Tx chain if needed.

- `ath_tx_paprd_cleanup`

This is a cleanup function which has the reverse effect `ath_tx_paprd_init()`. It frees up all the resources that the module used during its lifetime.

### 17.1.2.8 Data Structure

Following fields of `ath_softc` struct is used by PAPRD module.

```

int8_t          sc_paprd_rate;           /* PAPRD TX rate */
int8_t          sc_paprd_enable;         /* PAPRD enabled */
int8_t          sc_paprd_done_chain_mask; /* PAPRD Done on chain num */
int8_t          sc_paprd_chain_num;       /* chain num - current PAPRD frame
to be sent*/
int8_t          sc_paprd_abort;          /* PAPRD abort */
int8_t          sc_paprd_failed_count;    /* PAPRD failed count */
int8_t          sc_paprd_retrain_count;   /* PAPRD re-train failed count */
int8_t          sc_paprd_cal_started;     /* PAPRD CAL started */
int16_t         sc_paprd_bufcount;        /* Total paprd buf need to send for
1 paprd CAL */
spinlock_t      sc_paprd_lock;
struct ath_txq  sc_paprdq;              /* tx q for PAPRD frame */
struct ath_descdma sc_paprddma;          /* PAPRD Tx descriptor */

```

```

ath_bufhead           sc_paprdbuf;          /* PAPRD free buffer */
u_int16_t             sc_paprd_lastchan_num; /* Last chan PAPRD Cal completed */
int8_t                sc_paprd_thermal_packet_not_sent; /* PAPRD thermal packet */

```

HAL\_CHANNEL struct: PAPRD state (calibration done, paprd CAL table is applied to hardware after reset) is maintained as part of HAL\_CHANNEL struct.

```

u_int8_t   paprd_done:1,    /* 1: PAPRD DONE, 0: PAPRD Cal not done */
paprd_table_write_done:1; /* 1: DONE, 0: Cal data write not done */

```

### 17.1.2.9 Debugging

PAPRD debug messages are logged under ‘ATH\_DEBUG\_CALIBRATE’. This debug flag can be enabled at run time as well as compile time (sample code is left commented in ath\_tx\_paprd\_init).

### 17.1.2.10 Future Enhancement

Currently PAPRD training is done once on a particular channel. Changes in temperature or noise might require re-calibration.

### 17.1.2.11 Known Issues

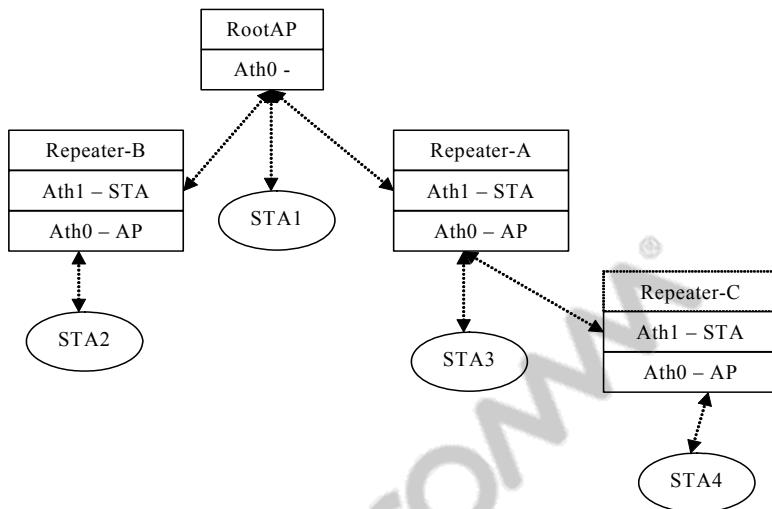
Before PAPRD training completes, Tx on PAPRD rates occurs at a lower power. Users might experience relatively low throughput initially and then see a boost in throughput. This window is 2 sec in STA mode and 200 ms in AP mode.

For a PAPRD training packet, TA, DA, RA can be any address. There is a possibility that if the air is sniffed during the PAPRD training, these packets will be seen on the air.

## 17.2 WDS

A Wireless Distribution System (WDS) is a system enabling the wireless interconnection of access points in an IEEE 802.11 network. It allows a wireless network to be expanded using multiple access points without the traditional requirement for a wired backbone to link them. The notable advantage of WDS over other solutions is it preserves the MAC addresses of client frames across links between access points. See [Figure 17-4](#).

WDS may also be considered a repeater mode because it appears to bridge and accept wireless clients at the same time (unlike traditional bridging). However, with this method, throughput is halved for all clients connected wirelessly.



**Figure 17-4 WDS Block Diagram**

### Terminology

- **Root AP**  
A Root AP is an AP with the WDS feature enabled, and which supports WDS 4-address communication. This AP will not have a STA VAP.
- **Repeater AP**  
A Repeater AP sends out beacons and supports IEEE802.11 association process which allows STA connecting to it. The STAs and PCs that associate/connect to the repeater could reach other STAs/PCs reachable using a Root AP through the WDS link.
- **WDS Node Table**  
A WDS Node Table is maintained in the Root AP that adds the MAC addresses of STAs (Ethernet or wireless) that are reachable through any repeater. Each WDS entry will have STA MAC address and the repeater AP node through which the STA can be reached.

### 17.2.1 Theory of Operation

The WDS design can summarized as follows:

- 4 address-frames are needed between the WDS links.
- When the Root AP WDS table is filled with WDS entries that includes following details.
  - a. MAC address of the STA
  - b. Node pointer of the Repeater through which STA can be reached.
  - c. MAC address of the Repeater through which STA can be reached.
  - d. Flags that contains details of STA. If STA is behind Root AP or behind the Repeater AP.

- When the Repeater AP WDS table is filled with WDS entries of STA MAC that are behind the Repeater AP. This is needed to detect the mcast echo.
- All the WDS table entries are periodically timed out to cleanup any stale entries.
- Roaming STA are handled by repeater AP sending l2uf to all, which makes the ROOT AP update its WDS table.

## 17.2.2 WDS Configuration Commands

### 17.2.2.1 Command Line

- Root AP

```
iwpriv ath0 wds 1
```

The above command sets the ath0 (AP VAP) to WDS capable.

- Repeater AP

```
iwpriv ath1 wds 1
```

The above command sets the ath1 (STA VAP) as WDS capable. The ath0 interface (AP VAP) need not be WDS enabled. Theoretically, if the AP VAP in the repeater VAP is WDS enabled, then another repeater can be chained to this repeater AP. Note that repeater chaining is not a tested feature. Hence, a repeater can associate only to a ROOT AP and not to another repeater AP.

- Wireless Extender AP

```
iwpriv ath0 wds 1
```

The above command sets ath0 (STA VAP) as WDS capable.

## 17.2.3 Direct Attach WDS Implementation

### 17.2.3.1 Build with WDS support

Ensure that UMAC\_SUPPORT\_WDS is defined while building the driver. On a Linux platform, it is defined in os/linux/include/osdep\_opts.h.

### 17.2.3.2 WDS API (UMAC Operations)

- **void ieee80211\_wds\_attach(struct ieee80211\_node\_table \*nt)**

When the node table is initialized, the function would be called for initialization of the WDS structure. This is not called per-vap.

- **struct ieee80211\_node \* ieee80211\_find\_wds\_node(struct ieee80211\_node\_table \*nt, const u\_int8\_t \*macaddr)**

The function is used to return the node through which macaddr can be reached. If no matching entry found NULL is returned.

- **Void wds\_update\_rootwds\_table(struct ieee80211\_node \* ni, struct ieee80211\_node\_table \*nt, wbuf\_t wbuf)**

When a 4-address frame is received by root AP, this function is called to update the WDS table with any needed entries.

Other WDS table-related node addition and deletion functions are defined in the ieee80211\_wds.c file. If WDS is not built, the above functions are defined as NULL ones.

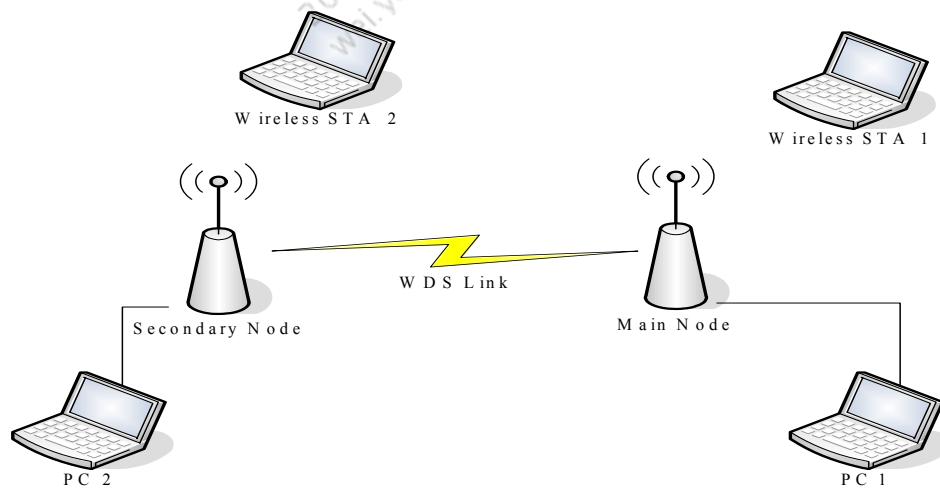
## 17.2.4 802.11ac Offload WDS Implementation

For an 802.11ac Offload solution, all the data packets are directly sent to the target and does not go through UMAC. The WDS table is completely maintained in the target.

The WDS configuration command is the same between direct attach and 11ac offload. The WDS enable/disable command is sent to the target firmware through the WMI command, WMI\_VDEV\_PARAM\_WDS.

## 17.2.5 NAWDS

The Non-Association WDS (NAWDS) is an AP feature that provides an alternative WDS solution that does not require any association between peer NAWDS APs. A NAWDS AP could be statically configured to receive WDS packets from and send packets to peer NAWDS APs. With the learning feature enabled, a NAWDS AP would learn the MAC address automatically to perform WDS. [Figure 17-5](#) is a typical NAWDS scenario where main node and secondary node is the NAWDS AP.



**Figure 17-5 Typical NAWDS Scenario**

### Terminology

- **NAWDS**

Non-Association WDS. In contrast with the association-based WDS, the NAWDS link is established without the necessity of going through the 802.11 association process.

- NAWDS AP

A NAWDS AP is an AP with the NAWDS feature enabled that supports WDS 4-address communication. Depending on sending a beacon or not, a NAWDS AP could be either a NAWDS Repeater or a NAWDS Bridge.

- NAWDS Repeater

A NAWDS Repeater sends out beacons and supports IEEE802.11 association process which allows STA connecting to it. The STAs and PCs which associate/connect to the repeater could reach other STAs/PCs in peer NAWDS AP through NAWDS link.

- NAWDS Bridge

A NAWDS bridge does NOT send out beacons. The PCs which connect to the bridge could reach other STAs/PCs in peer NAWDS AP through NAWDS link.

- Static NAWDS Repeater/Bridge

A static NAWDS AP does NOT have learning function. Users have to manually add the MAC address of a peer NAWDS AP into its NAWDS MAC table so that it can communicate with other NAWDS APs. If the MAC address of a NAWDS AP is not in the NAWDS MAC table, a static NAWDS AP would drop packets from that particular NAWDS AP.

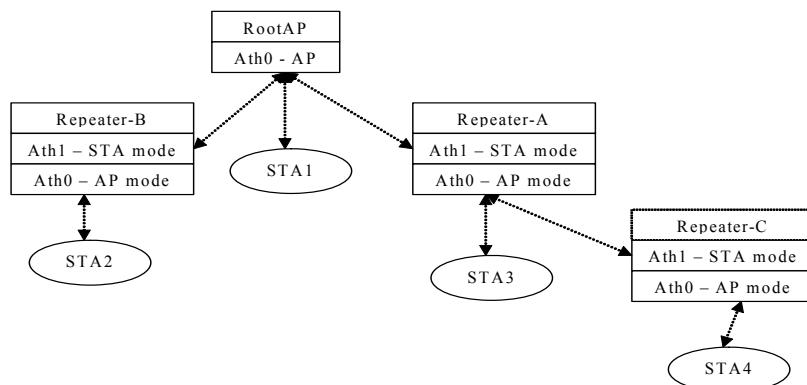
- Learning NAWDS Repeater/Bridge

A learning NAWDS AP has a learning function. Whenever it receives packets from peer NAWDS AP, it would learn the MAC address of the peer AP and add the address to its NAWDS MAC table so that it could further communicate with the particular AP through NAWDS link.

### 17.2.5.1 Theory of Operation

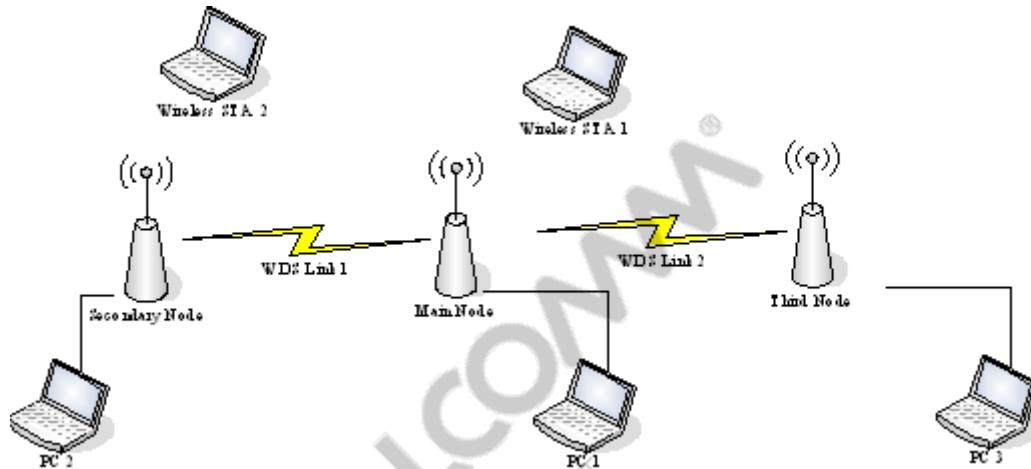
#### WDS to NAWDS

The existing WDS solution builds a WDS tree to forward frames and is association-based. The original WDS design could be depicted as in [Figure 17-6](#).



**Figure 17-6 Typical WDS Block Diagram**

NAWDS is more like a flat structure which means that any NAWDS node directly talks to one or multiple NAWDS peers. Users could either configure the MAC addresses of the peer NAWDS APs for each NAWDS AP or enable the learning feature to allow the NAWDS AP to add the MAC addresses of the peer NAWDS AP automatically when it get packets from peer APs.



**Figure 17-7 NAWDS Connections**

Due to the nature of NAWDS, some constraints are applied to the feature:

- It is necessary to have at least one static node. Other nodes know peer NAWDS nodes either by learning or by statically configuring.
- Between any two NAWDS nodes, frames must use four addresses for proper forwarding.
- All the NAWDS nodes in the network must have the same encryption setting. Currently only Open/WEP mode is supported.
- Since there is no association process, the capability of peer NAWDS AP must be configured in advanced.

### 17.2.5.2 Configuration

#### Configuration Descriptions

- wlanconfig athX nawds mode (0-4)

Configure the mode in which NAWDS AP is operating. The value is defined as below:

|          | Repeater<br>(Sending beacons) | Bridge<br>(NOT sending beacons) |
|----------|-------------------------------|---------------------------------|
| Static   | NAWDS MODE: 1                 | NAWDS MODE: 2                   |
| Learning | NAWDS MODE: 3                 | NAWDS MODE: 4                   |

NAWDS mode 0 means NAWDS DISABLED. Whenever mode is changed, the NAWDS MAC table would be cleared.

- wlanconfig athX nawds defcaps CAPS

When a NAWDS AP is operating in learning mode, it has no idea which capability the learned peer NAWDS AP has. In this situation, defcaps would be used. The CAPS is defined as below:

```
/* Bits 0 - 7 NSS */
#define NAWDS_REPEATERO_CAP_DS          0x01
#define NAWDS_REPEATERO_CAP_TS          0x02
#define NAWDS_REPEATERO_CAP_4S          0x04

/* Bits 8 - 15 CHWIDHT/HTMODE */
#define NAWDS_REPEATERO_CAP_HT20         0x0100
#define NAWDS_REPEATERO_CAP_HT2040       0x0200
/* VHT Capability */
#define NAWDS_REPEATERO_CAP_11ACVHT20    0x0400
#define NAWDS_REPEATERO_CAP_11ACVHT40    0x0800
#define NAWDS_REPEATERO_CAP_11ACVHT80    0x1000
#define NAWDS_REPEATERO_CAP_11ACVHT80_80 0x2000
#define NAWDS_REPEATERO_CAP_11ACVHT160   0x4000

/* Bits 16 - 23 TX BF */
#define NAWDS_REPEATERO_CAP_TXBF        0x010000
```

If CAPS equals 0, the HT rate would be disabled. To enable NAWDS\_REPEATERO\_CAP\_DS, at least one of NAWDS\_REPEATERO\_CAP\_HT20 and NAWDS\_REPEATERO\_CAP\_HT2040 must be specified.

- wlanconfig athX nawds override (0-1)

When running out of entries in the NAWDS MAC table (either by configuring too many NAWDS AP or by learning too many AP due to learning feature), enabling override would make NAWDS delete the MAC address which is occupied by a dead NAWDS AP.

If the override is disabled, no more MAC address could be added in NAWDS table when the table is full.

- wlanconfig athX nawds add-repeater MAC CAPS

Add a NAWDS AP with the specified MAC address and capability. The definition of CAPS is the same as the CAPS mentioned above.

- wlanconfig athX nawds del-repeater MAC

Delete a NAWDS AP with the specified MAC address

- wlanconfig athX nawds list

Display current NAWDS configurations

## Configuration Examples

- Static bridge and peer node supports HT20 rates

```
cfg -a AP_PRIMARY_CH=149
cfg -a AP_CHMODE=11NAHT40PLUS
cfg -a AP_SSID=edward-test
apup
iwpriv ath0 wds 1
wlanconfig ath0 nawds mode 2
wlanconfig ath0 nawds add-repeater 00:03:7F:0C:E4:B6 0x0100
```

- Learning bridge and by default peer NAWDS AP supports HT40/DS rates

```
cfg -a AP_PRIMARY_CH=149
cfg -a AP_CHMODE=11NAHT40PLUS
cfg -a AP_SSID=edward-test
apup
iwpriv ath0 wds 1
wlanconfig ath0 nawds mode 4
wlanconfig ath0 nawds defcaps 0x0201
```

## Configuration Interface for Linux

NAWDS could either leverage interface ioctl or iwpriv ioctl for its configuration. However, due to the shortage of iwpriv ioctl, NAWDS uses interface ioctl as its kernel-user space configuration interface. To further prevent running out of interface ioctl, a generic ioctl number IEEE80211\_IOCTL\_CONFIG\_GENERIC has been introduced to adopt various type of configuration in the future.

IEEE80211\_IOCTL\_CONFIG\_GENERIC takes a structure which contains a sub-command field and NAWDS defines the following sub-commands:

```
IEEE80211_WLANCONFIG_NAWDS_SET_MODE,
IEEE80211_WLANCONFIG_NAWDS_SET_DEFCAPS,
IEEE80211_WLANCONFIG_NAWDS_SET_OVERRIDE,
IEEE80211_WLANCONFIG_NAWDS_SET_ADDR,
IEEE80211_WLANCONFIG_NAWDS_CLR_ADDR,
IEEE80211_WLANCONFIG_NAWDS_GET
```

### 17.2.5.3 Implementation

#### MLME Operations

Currently the MLME functions shown below are exported for configuration usage.

##### Configure NAWDS peer repeater

```
@param    vaphandle: handle to the vap
@param    macaddr   : mac address for the peer NAWDS repeater
@param    flags      : capability of the peer NAWDS repeater
int wlan_nawds_config_mac(wlan_if_t vaphandle, char *macaddr, char
flags);
```

##### Delete NAWDS peer repeater

```
@param    vaphandle: handle to the vap
@param    macaddr   : mac address for the peer NAWDS repeater
int wlan_nawds_delete_mac(wlan_if_t vaphandle, char *macaddr);
```

##### Get configuration of a specific NAWDS peer repeater

```
@param    vaphandle: handle to the vap
@param    num       : index of the specific NAWDS repeater
@param    macaddr   : buffer to store the mac address of the peer NAWDS
repeater
@param    flags     : buffer to store capability flag of the peer NAWDS
repeater
int wlan_nawds_get_mac(wlan_if_t vaphandle, int num, char *macaddr, char
*flags);
```

##### Set NAWDS related parameters

```
@param    vaphandle: handle to the vap
@param    param     : specify which parameter to set
@param    val       : new NAWDS parameter passed as a pointer

int wlan_nawds_set_param(wlan_if_t vaphandle, enum ieee80211_nawds_param
param, void *val);
```

##### Get NAWDS related parameters

```
@param    vaphandle: handle to the vap
@param    param     : specify which parameter to get
@param    val       : buffer to store current NAWDS parameter

int wlan_nawds_get_param(wlan_if_t vaphandle, enum ieee80211_nawds_param
param, void *val);
```

The related NAWDS MLME parameters are defined as below:

```
enum ieee80211_nawds_param {
    IEEE80211_NAWDS_PARAM_NUM = 0,
    IEEE80211_NAWDS_PARAM_MODE,
    IEEE80211_NAWDS_PARAM_DEFCAPS,
    IEEE80211_NAWDS_PARAM_OVERRIDE,
};
```

## UMAC Operations

To avoid using too many #ifdef directives and exposing too much NAWDS detail to other parts of UMAC, 5 functions are exported to UMAC layer.

```
void ieee80211_nawds_attach(struct ieee80211vap *vap);
```

When a VAP is created, the function is called for initialization of NAWDS structure.

```
int ieee80211_nawds_send_wbuf(struct ieee80211vap *vap, wbuf_t wbuf);
```

The function is used to propagate packet among NAWDS peer AP in the UMAC Tx path.

```
int ieee80211_nawds_disable_beacon(struct ieee80211vap *vap);
```

The function is used to verify whether a beacon is required to be sent. For a NAWDS bridge, beacon is disabled.

```
int ieee80211_nawds_enable_learning(struct ieee80211vap *vap);
```

The function is used to verify whether NAWDS learning feature is enabled. If enabled, NAWDS would start the learning process when a 4 address-frame is received.

```
void ieee80211_nawds_learn(struct ieee80211vap *vap, u_int8_t *mac);
```

Do the actual learning and create a NAWDS peer node for NAWDS communication.

If NAWDS is not built, the aforementioned functions are defined as NULL ones.

## Build with NAWDS support

Ensure that UMAC\_SUPPORT\_NAWDS is defined while building the driver. On Linux platform, it is defined in os/linux/include/osdep\_opts.h. To change the number of NAWDS AP supported, modify UMAC\_MAX\_NAWDS\_REPEATERS in umac/include/ieee80211\_options.h.

## 17.3 Extender AP

This section outlines the design parameters and implementation for the extender-AP project. It provides the information, techniques, and procedures required to bring up and run the Qualcomm driver on a specific Linux based platform.

Extender-AP is a feature in AP, where AP act as a station (Client-AP) and forwards all three address data frames from the wireless interface to the device connected to the LAN and WAN port of the AP and vice versa. This functionality could be achieved by the existing implementation of the WDS client feature in the AP. But the AP to which the Client-AP is connected needs to understand a four-address frame. Implementing the three-address frame forwarding feature in AP removes this restriction, and allows the client-AP to connect with any legacy AP that does not support four-address frame format.

### 17.3.1 Configuration

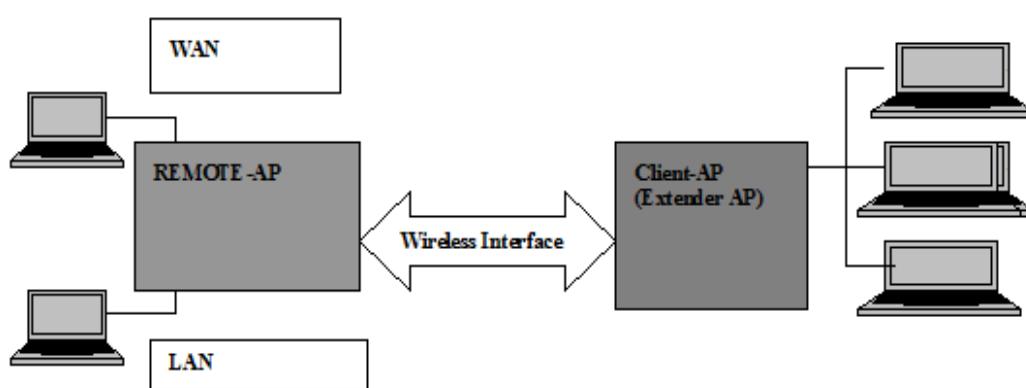
In the three-address WDS, two APs are used. One AP is brought up as a Normal AP by configuring the AP\_STARTMODE as standard (cfg -a AP\_STARTMODE = standard) and the other AP Extender AP is either brought up in AP mode or STA mode by configuring the AP\_STARTMODE as follows.

To bring up the AP in Extender AP/Station mode,

```
cfg -a AP_STARTMODE=extap (Extender-AP)
(or)
cfg -a AP_STARTMODE=extsta (Extender-STA)
```

There is no special configuration required except the start mode. For example the security configuration are same as normal AP.

### 17.3.2 Feature Description



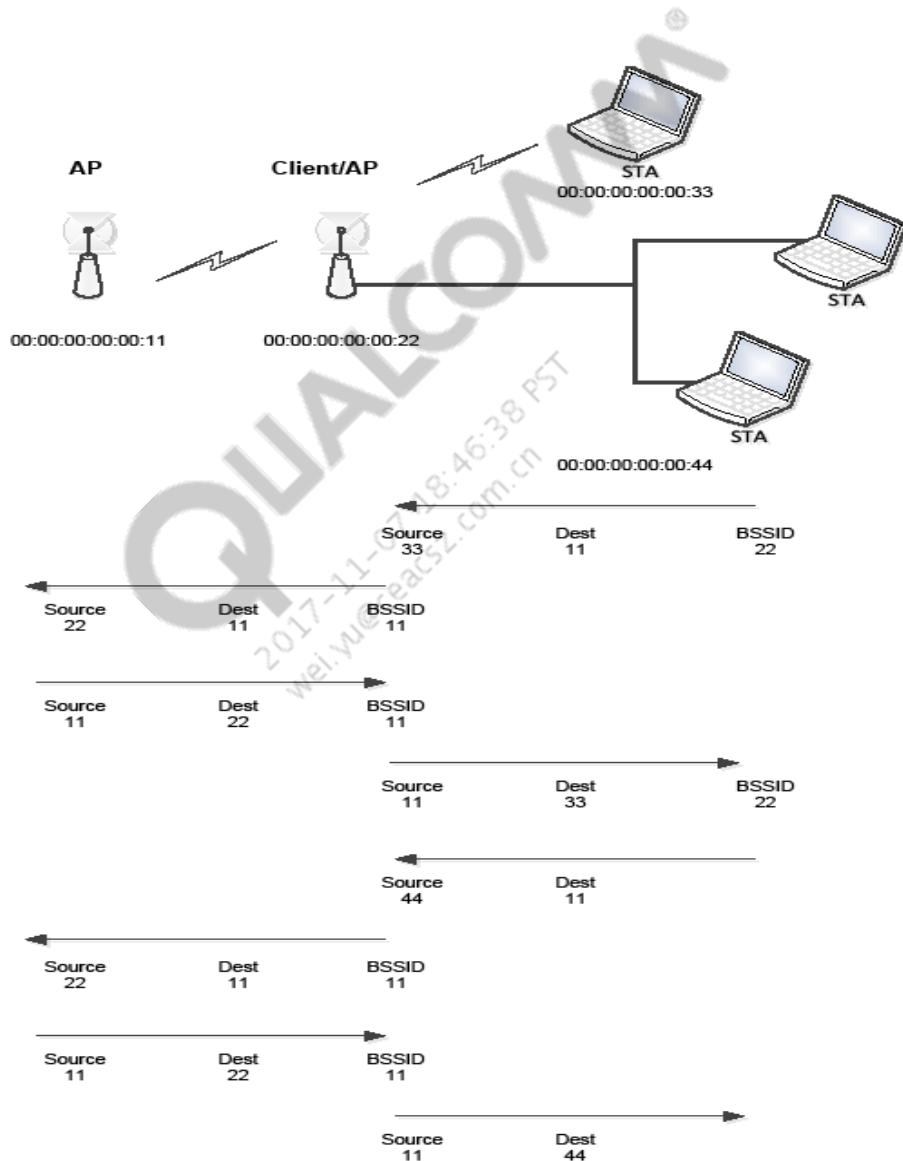
**Figure 17-8 Extender AP Block Diagram**

The following are the functional requirements

1. AP should be able to configure in Extender-AP mode through configuration parameters.

2. AP operating in Extender-AP mode (Client-AP) should dynamically learn the MAC and IPv4/IPv6 address of the end device connected to the WAN and LAN port of the AP.
3. AP operating in Extender-AP mode (Client-AP) should bridge the frames received from the end device connected to the LAN/WAN port of the AP to the AP through wireless interface and vice versa using the three-address format.

**Figure 17-9** explains a use case. Though the requirement is limited to wired clients, the design considers supporting a wireless client as well, so the diagram shows the wireless client also.



**Figure 17-9 Extender AP Use Case**

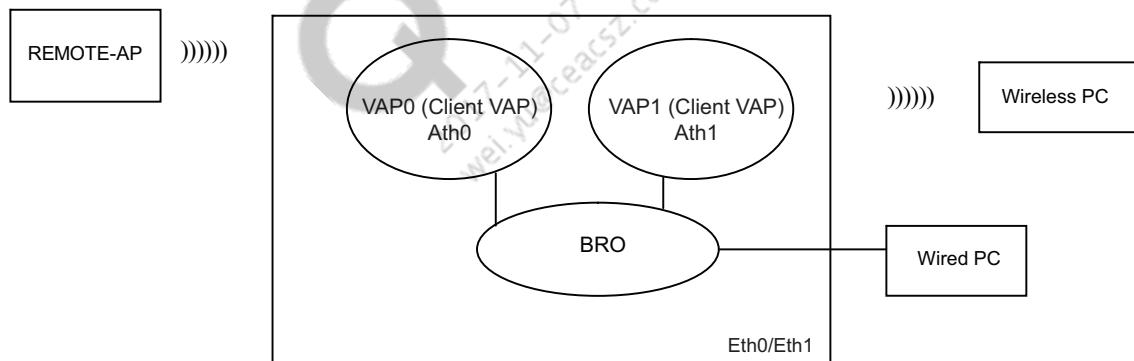
### 17.3.3 AP Configuration Mode

A new operating mode called “extender-ap” is defined to meet the above requirements. By setting the “AP\_STARTMODE” environment variable to “extender-ap” sets this operating mode. To support wireless end devices, a new environment variable called “EXT\_AP\_WIRELESS\_EUD” is defined. By default, wireless end devices are not supported. Setting this to YES enables the wireless end devices.

In the case of wireless end devices, Extender AP has 2 VAPS. One acts as the AP, and the other one acts as the client. The AP VAP allows clients to associate to itself. The Client-VAP associates with the remote AP and bridges the wireless/wired frames using 3 addresses to the remote AP. In the case of wired only end devices, the Extender AP has only the VAP, the client VAP.

Extender AP has the following interfaces:

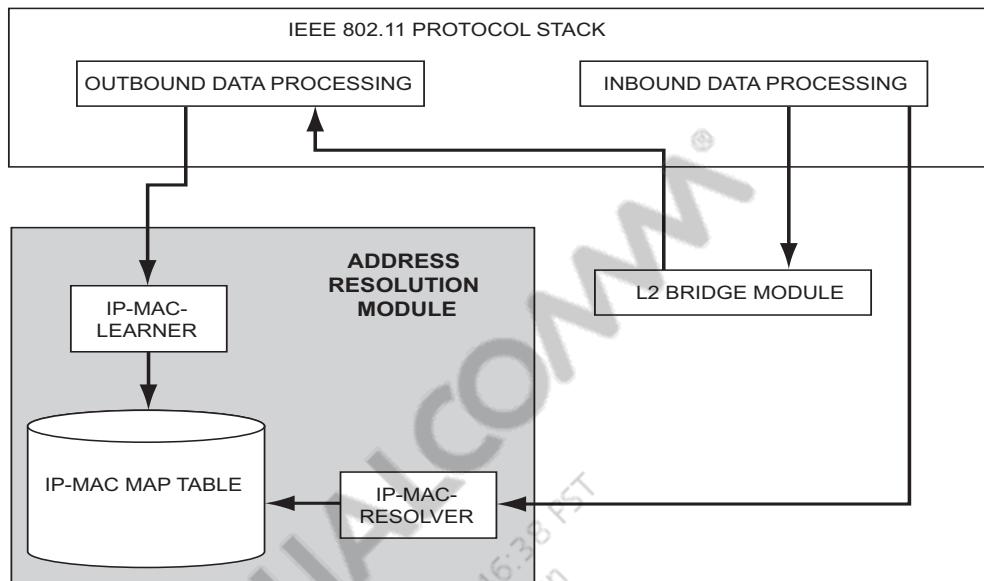
- Ath0 – Wireless interface for transmitting and receiving the packets to and from the remote AP
- Ath1 – Wireless interface for transmitting and receiving the packets to and from the wireless station connected to the Extended AP.
- Eth0 – LAN Ethernet interface for transmitting and receiving the packets to and from the wired stations connected to the Extender AP.
- BRO – Layer 2 Ethernet bridging interface to bridge the traffic between ath0, ath1 and eth0 interfaces.



**Figure 17-10 Extender AP Configuration Mode**

### 17.3.4 Top Level Design Overview

The following diagram identifies the major feature components, and their relationship to each other and the system overall. This provides the highest level of design description.



**Figure 17-11 AP Extender Overview**

#### 17.3.4.1 Functional Design

When an AP is configured in Extender-AP mode, the client VAP node has a special flag called “ext-ap-3addr” is set to 1 for indication.

#### 17.3.4.2 IEEE 802.11 Protocol Stack

##### Outbound Data Processing

This does outbound data frame processing. It receives the Ethernet frames from the bridge control module, and encapsulates the 802.11 header before passing it to LMAC layer. If the node object ext-ap-3addr flag is set, then it will call the address resolution module IP-MAC-LEARNER routine to automatically learn the IPV4/IPv6 address and MAC address, and also replace the ARP source MAC address to its own MAC address.

##### Inbound Data Processing

This routine does inbound data frame processing. It receives the 802.11 frames, decapsulates the MAC frame, and passes the Ethernet frame to Bridge control interface. If the node object ext-ap-3addr flag is set, then it will call address resolution module IP-MAC-RESOLVER routine to get the destination MAC address for the inbound frame.

### 17.3.4.3 Address Resolution Module

#### IP-MAC Learner

This learns the MAC, IPV4, and IPV6 addresses of the wired and wireless stations updates them in the IP-MAC mapping table. It looks for a IPV4 ARP reply message, decodes the IPV4 address and MAC address, and updates the IPV4 and MAC in the IP-MAC map table. It also looks for a IPV6 ICMP node discovery message, decodes the IPV6 address and MAC address, and updates them in the IP-MAC map table.

#### IP-MAC Map Table

This maintains the IP-MAC map table. Each entry in this table consists of IPV4/IPV6 and MAC addresses, and a key index is IPV4/IPV6 address. It provides the access routine for creating an entry in the table, deleting an entry, modifying an entry and also finds the MAC address for an IPV4/IPV6 address.

#### IP-MAC Resolver

This resolves the IPv4, IPv6 address to MAC address for the inbound data frames.

### 17.3.5 Frame processing

This section explains changes required in the inbound and outbound frame processing of the Extender-AP client VAP.

#### 17.3.5.1 Inbound processing

Strip the 802.11 Mac header from the received frame and create Ethernet header from 802.11Mac header as follows:

- If the destination MAC address is broadcast or multicast then create the Ethernet header with same destination MAC address as received.
- If the destination MAC address is unicast and the packet type is IP, then get the IPV4 or IPV6 address from the IP packet and resolve the IP address to MAC address using IP-MAC resolver. Set this address as the destination MAC address in the Ethernet header. If it can't be resolved then broadcast the frame.
- If the destination MAC address is unicast and packet type is ARP, then get the Target MAC address from the ARP payload and set it as Destination MAC address in the Ethernet header.
- If the destination MAC address is unicast and packet type IP and DHCP, and if the message is a DHCP OFFER message, then get the client HARDWARE address from the DHCP offer message and set it as Destination MAC address in the Ethernet header.

#### 17.3.5.2 Outbound processing

Before encapsulating the 802.11 MAC header from the Ethernet header, automatically learn the IPV4/IPV6 address and its MAC address from the frame and update in the IP-MAC map table as follows.

- If the Ethernet packet type is ARP, then get the sender MAC address and sender IPV4 address from the ARP table and update it in the IP-MAC map table.
- If the Ethernet packet type is IP and IP protocol is ICMP, and it is a ICMP node discovery message, get the IPV6 address from the ICMP payload and the sender MAC address from the Ethernet and update it in the IP-MAC map table.

### 17.3.6 Implementation

This section provides details for topics presented in the top-level design section.

#### 17.3.6.1 UMAC Operations

- static mi\_node\_t mi\_tbl[NUM\_MITBL\_ENTRIES];  
When the node table is initialized, the function would be called for WDS structure initialization. This is not called per-vap.
- ieee80211\_mitbl.c  
The file implements all mi table related functions.
- ieee80211\_extap\_input()  
This function is the interface point for Rx packets when extAP is enabled in the STA VAP.
- ieee80211\_extap\_output()  
This function is the interface point for any Tx packets to be sent from the STA VAP (extender AP) to the connected AP.
- ieee80211\_extap\_out\_ipv4(), ieee80211\_extap\_out\_arp(), ieee80211\_extap\_in\_ipv6(), ieee80211\_extap\_out\_ipv6()  
The various extap functions that deal with Rx or Tx packets of ipv4, ipv6 or ARP packets. These functions are responsible for MAC address translation from the table and also for updating of the MI table. They also respond to ARP.
- MI\_TABLE\_AS\_ARRAY, MI\_TABLE\_AS\_TREE  
The macros define if the MI Table is implemented as an array or as a binary tree. It is defined based on the performance requirement.

#### 17.3.6.2 Build with extAP support

Ensure that ATH\_EXT\_AP is defined while building the driver.

## 17.4 Proxy STA

Proxy STA is an AP mode that allows wireless-to-Ethernet bridging, extending the wireless network to wired clients such as a set top boxes, game consoles, or a room of Ethernet-networked computers, which are not equipped with WLAN interfaces originally.

### 17.4.1 ProxySTA vs. WDS

The following are advantages of ProxySTA compared to 4A:

1. Compatibility with third-party devices (peer WLAN device is not aware of the proxy feature, it sees the Ethernet STA as a unique WLAN STA)
2. Because ProxySTA uses 3-addresses; it is simpler to configure and provides for better performance.

The disadvantages of ProxySTA are as follows:

1. Proprietary solution
2. No range extension
3. Ethernet-to-wireless only

### 17.4.2 Terminology

- Azimuth Mode: A MAC hardware mode to support ProxySTA
- DDD: Decouple Decryption Disabled
- DDE: Decouple Decryption Enabled
- MAT: MAC Address Translation

### 17.4.3 Theory of Design

When the AP is configured in the ProxySTA mode, a master VAP is created and used to access, configure, and terminate traffic meant for the AP. For each Ethernet client connected to the AP, a Proxy STA VAP is created. This VAP within the ProxySTA domain is an IEEE 802.11 STA associated with the AP. The ProxySTA VAPs have a 1:1 relationship with the Ethernet clients connected through the Ethernet port of the Access Point. With some modification for the bridge code to support ProxySTA (discussed later), each Ethernet client is able to communicate inside the wireless network as if it was really associated with the AP directly. In the ProxySTA architecture, the understanding of ProxySTA mode is within the Access Point only. Other partners in this picture including the AP and its associated STAs are not aware of the ProxySTA mode.

### 17.4.4 MAC Address Translation

The basic mechanism involves translating the MAC address of one or more Ethernet-connected clients to one or more address that falls within a range for which the chip can be programmed to ACK. This procedure relies on the “ADDR1 Mask”. This mask is widely used today to enable

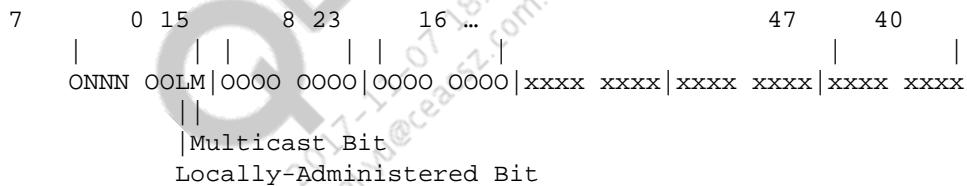
multiple VAP operation for APs. The mask is applied to the ADDR1 (RA) address of a received frame to determine whether a correctly-received frame should cause the chip to generate an acknowledgement (ACK or BA). This address range is referred to as the Acknowledgement Address Range (AAR). Frames transmitted by an Ethernet client will have their Original MAC Address (OMA) systematically substituted for a Virtual MAC Address (VMA) which falls within the AAR when that frame is transmitted over WLAN to the AP.

In addition to MAC address translation, certain “inter-layer” protocols which embed a MAC address into the Ethernet frame payload need to be handled—for example, ARP and ICMPv6-NDP. Magic Packet™ is out of the scope of this implementation. Translation required for these protocols is detailed in individual sections that follow. VMA “collision” is minimized through the allocation procedure described in section [17.4.4.1](#).

#### 17.4.4.1 Interfaces and VMA Allocation

Each Proxy STA is represented on the device by an interface called a “VAP” (Virtual AP). When an Ethernet client is detected, software creates a STA “VAP,” which proxies the Ethernet interface.

The new VAP is created, passing the MAC address of the Ethernet client (OMA). As part of VAP creation, the OMA is stored and a new VMA is allocated and used as the WLAN MAC address of the VAP. The following illustrates the MAC-48 address space, where O is a bit in the 3-byte Organizational-Unique Identifier (OUI), and x is a bit in the 3-byte NIC-specific region of the MAC address.



To create a VMA, the Locally-Administered (LA) bit of the WLAN client MAC address is set, and bits NNN are set to represent the index of the VMA. The first VMA allocated has an index of 0, the second an index of 1, and so on. Up to 8 VMAs can be allocated using this method. Since the NIC-specific octets remain unchanged, the probability of collision is minimized. Only other devices with the same NIC-specific octets also making use of a similar LA bit based addressing scheme would have the same MAC address, which is highly unlikely in practice.

The following shows one scenario for derived VMAs where the WLAN STA MAC address is 00:03:7f:c0:ff:ee. The Address1 Mask that the chip uses to determine whether to acknowledge frames is also detailed.

|             | Original MAC      | Virtual MAC       | Address1 Mask     |
|-------------|-------------------|-------------------|-------------------|
| WLAN STA    | 00:03:7f:c0:ff:ee | n/a               | ff:ff:ff:ff:ff:ff |
| Proxy STA 0 | 00:1f:f3:aa:bb:cc | 02:03:7f:c0:ff:ee | fd:ff:ff:ff:ff:ff |
| Proxy STA 1 | 00:1f:f3:dd:ee:ff | 12:03:7f:c0:ff:ee | ed:ff:ff:ff:ff:ff |
| Proxy STA 2 | 00:1f:f3:01:23:45 | 22:03:7f:c0:ff:ee | cd:ff:ff:ff:ff:ff |

#### 17.4.4.2 Key Exchange

When a STA “VAP” is created, the creator needs to be informed of the VMA that is allocated on behalf of the Ethernet client. The VMA is required by the supplicant for key exchange, since the MAC address is used by the supplicant and the authenticator in key mixing.

Refer to “[Configuration Tools Support for ProxySTA](#)” for the method for providing this address to the supplicant on the device.

#### 17.4.4.3 General Transmit and Receive of data frames

All link-layer frames to be transmitted over the WLAN whose source MAC address matches the OMA shall have their source MAC address replaced by the VMA. This can be done either before (while the frame is still in 802.3 or Ethernet II format) or after transcapsulation by the driver (that is, after conversion to 802.11 format). Note that in each case, different driver layers are modified.

All link-layer unicast frames received from the WLAN whose destination MAC address matches the VMA shall have their destination MAC address replaced by the OMA. Again, replacement can be performed either before or after transcapsulation.

#### 17.4.4.4 Protocol Issues

Any protocol in which a client embeds its own MAC address within the “data” payload (non-header portion) of a frame poses a potential issue. In general, any and all protocol spoofing will occur prior to transcapsulation (substituting the Ethernet MAC header for the 802.11 MAC header) on uplink, or after transcapsulation on downlink. In other words, the detection and possible substitution of MAC address within the data payload is performed on Ethernet-formatted frames.

To keep the implementation simple, protocol transcapsulation is done for 802.3 frames on both sides.

##### ARP

Since ARP replies sent from an Ethernet client will contain the OMA in the data payload, the driver will need to look for these ARP packets coming from the bridge, and substitute the OMA with the appropriate VMA. Likewise, ARP packets received from the WLAN are also inspected, substituting the VMA for the OMA, as appropriate. ARP uses EtherType 0x0806. [Figure 17-12](#) shows the generic format of an ARP packet.

| bit offset | 0 - 7   | 8 - 15                 | 16 - 31                                       |
|------------|---|------------------------|---|
| 0          | Hardware type (HTYPE)                         |                        | Protocol type (PTYPE)                         |
| 32         | Hardware length (HLEN)                        | Protocol length (PLEN) | Operation (OPER)                              |
| 64         | Sender hardware address (SHA) (first 32 bits) |                        |   |
| 96         | Sender hardware address (SHA) (last 16 bits)  |                        | Sender protocol address (SPA) (first 16 bits) |
| 128        | Sender protocol address (SPA) (last 16 bits)  |                        | Target hardware address (THA) (first 16 bits) |
| 160        | Target hardware address (THA) (last 32 bits)  |                        |   |
| 192        | Target protocol address (TPA)                 |                        |   |

**Figure 17-12 ARP Packet Format**

For all frames with an EtherType of 0x0806 to be transmitted over WLAN, and meeting the following criteria, the SHA is replaced with the VMA corresponding to the OMA. All other fields remain unchanged.

- HTYPE == 0x0001 (Ethernet)
- PTYPE == 0x0800 (IP)
- HLEN == 6 (Ethernet MAC address is 6 bytes)
- PLEN == 4 (IP address is 4 bytes)
- OPER == Any (to cover replies, gratuitous ARP, ARP probes, and possible RARP usage)
- SHA matches the OMA

For all frames with an EtherType of 0x0806 received from WLAN, and meeting the following criteria, the THA is replaced with the OMA corresponding to the VMA. All other fields remain unchanged.

- HTYPE == 0x0001 (Ethernet)
- PTYPE == 0x0800 (IP)
- HLEN == 6 (Ethernet MAC address is 6 bytes)
- PLEN == 4 (IP address is 4 bytes)
- OPER == Any (to cover replies, gratuitous ARP, ARP probes, and possible RARP usage)
- THA matches the VMA address

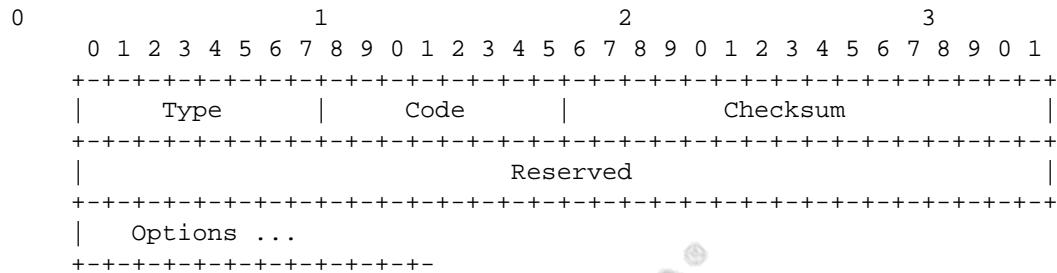
## Magic Packet

Magic Packet translation is not supported in this version.

## ICMPv6 Neighbor Discovery Protocol

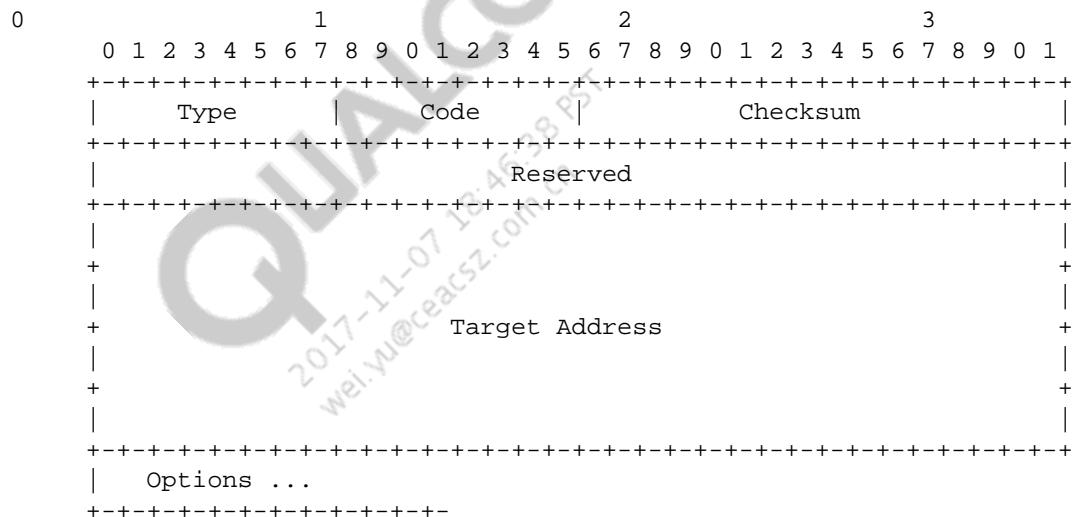
- Router Solicitation Message Format

Hosts send Router Solicitations in order to prompt routers to generate Router Advertisements quickly.



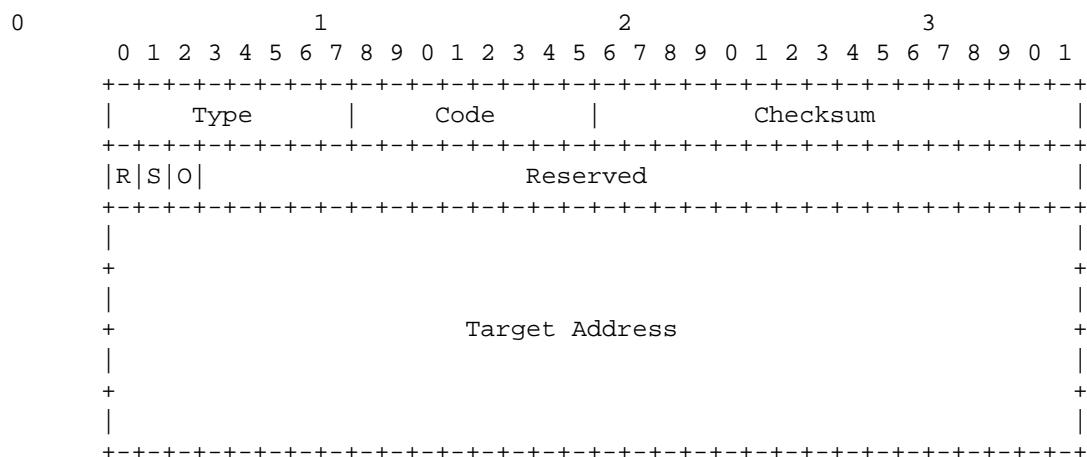
#### ■ Neighbor Solicitation Message Format

Nodes send Neighbor Solicitations to request the link-layer address of a target node while also providing their own link-layer address to the target. Neighbor Solicitations are multicast when the node needs to resolve an address and unicast when the node seeks to verify the “reachability” of a neighbor.



#### ■ Neighbor Advertisement Message Format

A node sends Neighbor Advertisements in response to Neighbor Solicitations and sends unsolicited Neighbor Advertisements to (unreliably) propagate new information quickly.



| Options ...

---

## Options field: Source/Target Link-layer Address

## Fields: Type

1 for Source Link-layer Address  
2 for Target Link-layer Address

The Target Link-layer Address is replaced with the VMA corresponding to the OMA. In addition, the ICMPv6 checksum, if included (non-zero) is appropriately updated. The IP header checksum need not be updated as it covers only the IP header, which is unmodified.

#### **17.4.4.5 Data Packet Encryption**

The usage of normal STA mode for the VAPs introduces a problem that is peculiar to this mode, and is not seen in the special Proxy-STA mode. The normal non-proxy mode of operation is described in this section.

When a key is negotiated for each STA with the AP, software will create an entry in a hardware key cache. This cache is a table with entries, each consisting of one or more keys used to encrypt and decrypt packets to a STA, and the corresponding peer address. When a packet is transmitted for any STA, software will provide the index of the key entry so that hardware can use it to encrypt the packet.

On receipt of a packet, the MAC hardware uses the transmitter address to lookup the corresponding key. If an AP sends a packet to a STA, the hardware searches for the AP MAC address in the key cache, and use the key corresponding to the address to decrypt the packet.

In proxy mode, all of the STAs being proxied connect to the same AP. A single entry is maintained for each STA in the key cache. Each entry has the same source address associated with it: that of the common AP. When a packet is received, hardware uses a linear search to look up the source address of the packet in the key cache. This lookup always results in the same cache entry, and implicitly, the same key being found. Because packets from the AP can be for any of the STAs being proxied, this methodology results in errors in the decrypt process when the key is mismatched. These errors are flagged as such by MAC hardware in the status field of the receive descriptor.

This problem is fixed by the following method. When the error in the descriptor is seen by the software function that is handling receive, the MAC address field of the wrong key cache entry for the packet is replaced by zeros so that the next time the key cache is looked up, this entry does not be located. The key cache entry containing the correct key is updated to hold the correct transmitter address, so that a subsequent look up of the key cache will result in the key from this modified entry being used for decryption. The packet that was decrypted incorrectly is re-encrypted with the incorrect key by software, and decrypted with the correct key (again by software), to yield the correct data. This ensures that no packets are dropped in this process. Note

that moving the address from one entry to the other is not atomic (two steps, zero first entry, move address to 2nd entry). This could result in packets received after zeroing out the first entry's address being uploaded by hardware without any decryption because the key is not found in the cache. Such packets do not need to be re-encrypted and a software decrypt is adequate.

This solution assumes that packets to and from any endpoint (STA) in a network are *bursty* in nature. Therefore, if the key for the first packet of a series of packets is incorrect and if software replaces the key quickly, subsequent packets are decrypted correctly by hardware. Packets that are decrypted incorrectly are rectified by software re-encrypt followed by decryption. This overhead is acceptable because this is a rare phenomenon.

However, if the aforementioned assumption does not hold true and packets are received alternately for multiple stations, this solution results in a doubling of software effort to re-encrypt and decrypt the packet.

The TCP iperf testing result for this MAT mode is approximately 2.5 times faster than a pure software cryptographic solution.

#### 17.4.5 Configuration Tools Support for ProxySTA

The wpa\_supplicant requires an understanding of the VMA, as it needs it to do key mixing and handles MLME/SME (if enabled).

The wpa\_supplicant is extended with a new option “-m” to specify the VMA before the key exchange. According to [Interfaces and VMA Allocation](#), the VMA for each interface can be predicted.

#### 17.4.6 Configure ProxySTA

To configure ProxySTA for the Qualcomm Technologies code base; that is, within the phoenix\_release branch, use the script “/etc/proxy-sta”:

1. Create the original station

```
/etc/proxysta ath0 0 <ssid>
```

(/etc/wpa\_supplicant.conf needs to be modified if WPA/WPA2 is used)

2. Create the first ProxySTA

```
/etc/proxysta ath0 1
```

3. Create the second ProxySTA

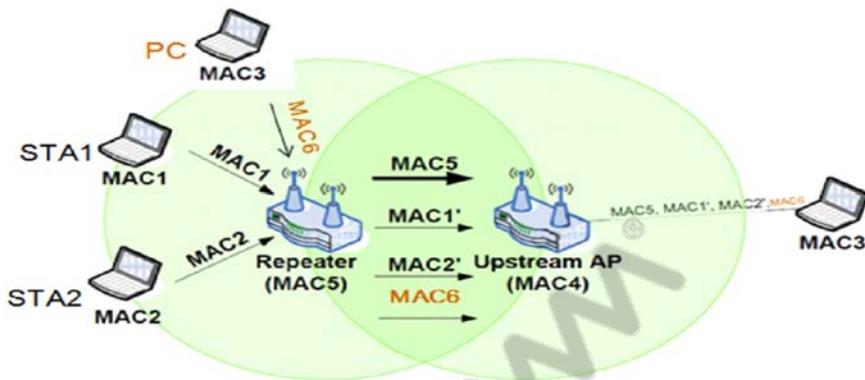
```
/etc/proxysta ath0 2
```

```
...
```

#### 17.4.7 Q-WRAP (Qualcomm Wireless Repeater AP)

Q-WRAP is the superset of Proxy STA. It supports both Ethernet clients as well as wireless clients. Wireless clients benefit from Q-WRAP through its range extension capability.

### 17.4.7.1 Design Overview



**Figure 17-13 Q-WRAP Overview**

- In Figure 17-13, MAC1 and MAC2 cannot associate to the upstream AP (or Root AP) because of the long distance. The Repeater (WRAP, MAC5) is used to relay traffic between MAC1/2 and Root AP. Because 4-address frames are not used, WRAP creates a separate virtual STA (or Proxy STA) for each client. Otherwise, if WRAP only creates a single Proxy STA, there is no way for WRAP to de-multiplex the downlink traffic from Root AP in Layer 2.
- WRAP operates as multiple independent STAs in the Root AP wireless network (N1). At the same time, WRAP also operates as an AP to associate all the long-range clients (MAC1/2) into its own wireless network (N2). WRAP operates on both N1 (Proxy STA side) and N2 (AP side) so that it can relay traffic between MAC1/2 and Root AP network (both wireless STAs; that is, MAC3 and the Distribution System).
- The single-radio WRAP operates on the same radio frequency to avoid channel change overhead and disadvantage (that is, frames lost during off channel). Thus, N1 and N2 share the same wireless medium. One restriction with this design is that WRAP cannot use the same client original MAC Address (OMA) to associate with the Root AP.
- The reason is that those frames originated from Root AP to Proxy STA (MAC1') might also be received by the client (MAC1). If the same MAC addresses are used, either the WRAP or client itself might ACK (or Block ACK) to the Root AP. This multi-ACK problem confuses the Root AP and can cause major receive unreliability on WRAP. To overcome this problem, Proxy STAs on WRAP use a Virtual MAC Address (VMA) to associate with the Root AP. VMA has 1:1 mapping to OMA on WRAP. The same mapping rule is used across all the WRAPs. The OMA/VMA mapping causes some issues for the upper layer protocols. [Section 17.4.4](#) describes this in detail.
- Some Qualcomm Technologies designs, such as AR934x and later support an “Azimuth Mode” that can be used to implement the Proxy STAs on WRAP. Software decryption on the WRAP AP side will be used for uplink traffic. A hardware change has been done in the QCA953x 2.0 chip to handle the decryption of packets in hardware for the uplink traffic. This essentially removes the need for software decryption and helps improve the throughput for uplink and bi-directional traffic.
- To support both Ethernet clients and wireless clients connected to WRAP, the Linux bridge code needs to be modified to support the special bridging functions.

### 17.4.7.2 Challenges for WRAP

#### Multi-ACK (or BA) problem

WRAP creates “proxy” STA VAPs on the WRAP to communicate with the Root AP on behalf of the real clients. For Ethernet clients, the same MAC address of the Ethernet client is used as the PSTA VAP MAC address. For WRAP, this mode will not work, since the PSTA VAP still uses the same MAC address of the wireless client to associate with the Root AP, when the Root AP sends a unicast frame to the PSTA VAP, the wireless client might also try to ACK (or BA) it. The reason is that the RA used by the AP matches both the address of the PSTA-VAP and the actual endpoint that is being proxied. Further, the moving nature of the wireless clients prevent us from physically positioning the wireless clients to a distant place where it does not receive any signals from the Root AP. To solve this problem, a different MAC address is used for PSTA VAP other than the wireless client and perform proper MAC address translation in the driver. See section [Section 17.4.4](#) for more details.

#### Hardware decryption

In normal mode (for both STA and AP), when Qualcomm Technologies hardware performs packet decryption, it uses A2 (transmitter address) of the received frame for key search (to locate the corresponding PTK for decryption). In Azimuth Mode, A1 (the receiver address) is used in the key search to support ProxySTA. However in WRAP mode, since both normal AP VAPs and PSTA VAPs for the same radio exist at the same time, the hardware will be either in Azimuth Mode or normal mode.

**NOTE** Throughput is less (~27 Mbps) in security mode (WPA2+PSK) compared to the open mode. There is a throughput difference of ~7 Mbps between the open and security mode in IPQ4019 due to the CPU getting loaded to 100%.

The following table lists the limits on maximum number of wired and wireless stations:

**Table 17-1 Maximum number of wired and wireless clients**

| Chipset model                          | Maximum number of clients (wireless + wired) | Total VAPs on QWRAP (PSTAs + MPSTA + AP VAP) |
|--|--|--|
| QCA9980                                | 22   | 24   |
| QCA9984                                | 28   | 30   |
| IPQ401x                                | 28   | 30   |
| QCA9886                                | 22   | 24   |
| QCA9889                                | 14   | 16   |
| Direct attach chipsets such as QCA9531 | 30   | 32   |

### 17.4.7.3 WRAP Design Target and Limitation

1. If maximal Ethernet/wireless clients is 3, requires 64 MB DRAM
2. Security mode: open, WPA2-PSK, and WEP
3. Enterprise authentication modes are not supported.
4. Roaming – supports STA roams among WRAPs but not between WRAP and root AP.
5. Other modes such as WDS are not supported when the Q-WRAP/proxy mode is enabled.  
Also, no dynamic switching between Proxy STA and other modes.
6. WPS is currently not supported.
7. For the list of protocols supported in payload when MAT is used, see the previous sections.
8. Isolation is supported only in single QCA9980 radio.
9. DBDC is supported as follows:
  - a. One radio as STA (connect to root AP) and the other radio as just AP
  - b. One radio as AP and STA (connect to root AP) and the other radio as just AP
10. Cascading Q-WRAP is supported only with unique MAC addresses defined in each Q-WRAP AP.

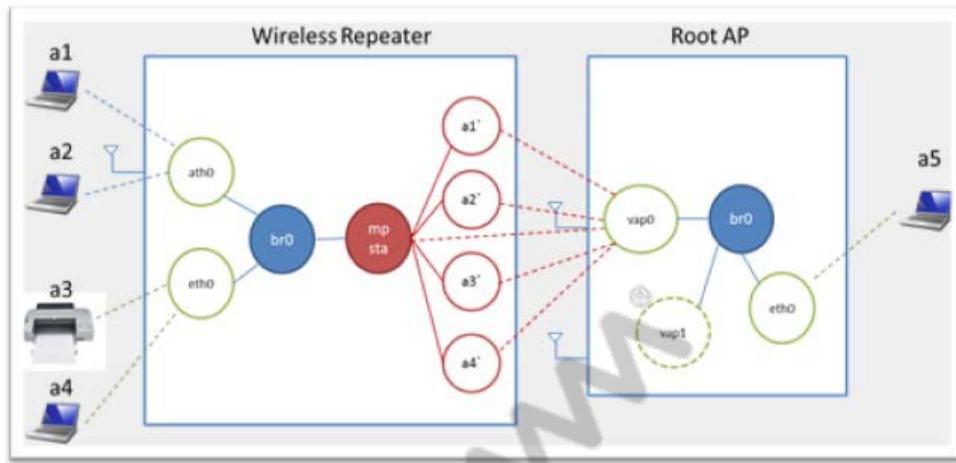
### 17.4.7.4 Q-WRAP L2 Bridging

#### Design

In [Figure 17-14](#), a1 and a2 represent wireless stations; a3 and a4 are wired stations that cannot connect directly to the RootAP. The Repeater AP is used to relay traffic between a1, a2, a3, a4, and Standard/RootAP.

The WRAP creates proxy-STA (virtual STA) for each client station. These are represented as a1', a2', a3' and a4' corresponding to the a1, a2, a3 and a4 stations respectively. These operate as multiple-independent STAs in the RootAP wireless network (vap0). The WRAP also operates as AP (ath0) to associate all long-range clients (a1, a2, a3, a4) into its wireless network.

The MP-STA (main proxy-STA) VAP represents the main repeater station, and implements all the WRAP-specific functionality (MAT, client-proxySTA mapping etc.). The MP-STA also connects to RootAP as a station, for management-related functionalities (key exchange etc.).



**Figure 17-14 Q-WRAP L2 bridging design**

### L2 bridging issue

The WRAP imposes the following challenges for L2 bridging implementation:

- The traditional Linux bridge does packet forwarding (i.e., egress port selection) based on the destination MAC address. For WRAP, packets originating from the long-range client needs to be sent to RootAP through the corresponding proxySTA. Hence, forwarding decisions to proxySTA ports (that is, uplink direction) need to be taken based on the client source MAC address and not the destination address.
- The traditional Linux bridge does port-learning from the source MAC address of the packets originating from an ingress port. In the downlink direction (RootAP to WRAP AP), it is possible for packets with the same source MAC address to arrive from different ingress proxySTA ports.

For example, a packet from a5 to a1 comes from the a1' port and a packet from a5 to a2 comes from the a2' port. Both these packets have “a5” source MAC address but come from two different ports. This may cause issues in bridging.

To solve these two issues, packet forwarding decisions for proxySTAs are handled in the wireless driver (by MP-STA). Only MP-STA is added as an interface to the bridge instead of all the proxySTA VAPs. All the proxySTA interfaces are collectively represented as MP-STA to the Linux network stack. The MP-STA maintains a list of all proxySTAs along with their addresses in the “proxySTA table” to aid in the packet forwarding decisions.

### MAC address translation

In order to use the ‘ADDR1 Mask’ hardware mechanism to filter the received packets and generate ACK/Interrupt, it is required that the MAC addresses for all long-range clients be translated to a common address pattern (mask) that could be programmed in the hardware. The mask is applied to the ADDR1 (RA) address of a received frame to determine whether a correctly-received frame should cause the hardware to generate an acknowledgment (ACK or BA).

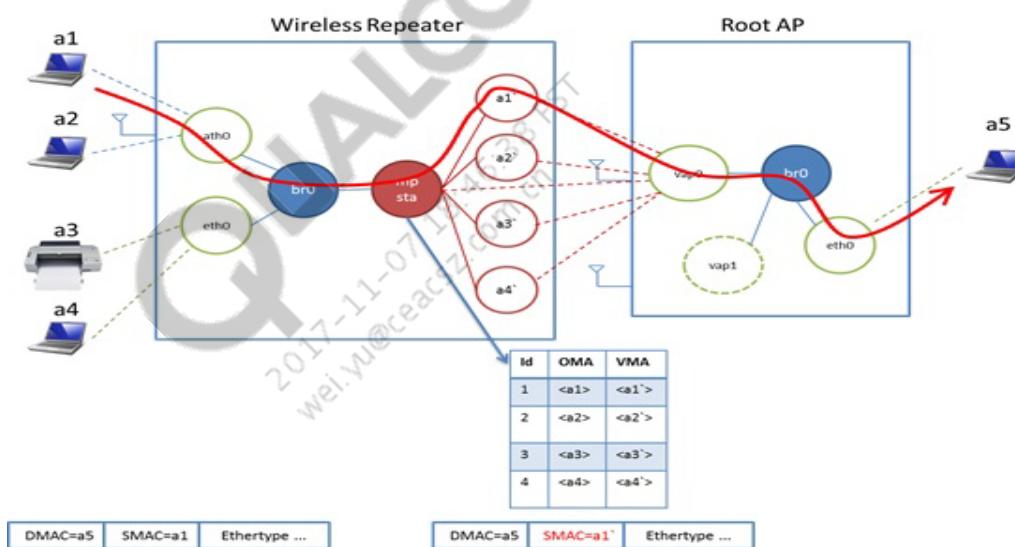
The frames transmitted by the long-range client will have their Original MAC Address (OMA) substituted for a Virtual MAC Address (VMA), which falls within the ADDR1 mask, when that

frame is transmitted over WRAP to the Root AP. The MP-STA VAP maintains the OMA- VMA mapping for all clients in the ‘proxySTA table’.

#### 17.4.7.5 Data Flow

##### Uplink

1. For transmitting a packet from ‘a1’ to ‘a5’, repeater AP ath0 receives the packet and hands it over to the Linux bridge. The normal mode of decryption (transmitter address is used for key search) is used. The bridge forwards the packet to the MP-STA VAP (port already learnt through ARP).
2. In IPQ806x, where the network subsystem (NSS) implements the fast path, packets are handed over to the NSS and not to the Linux bridge. NSS forwards the packet to the MP-STA VAP (through Wi-Fi redirect path), as the bridge does in the above step. Hence, the data path in the case of NSS is exactly the same as shown in [Figure 17-15](#), except that NSS replaces “br0”.



**Figure 17-15 Uplink – Non-Isolation**

3. When the packet reaches MP-STA (hard\_start\_xmit), VMA needs to be looked-up in the proxySTA table with the source MAC address as key.

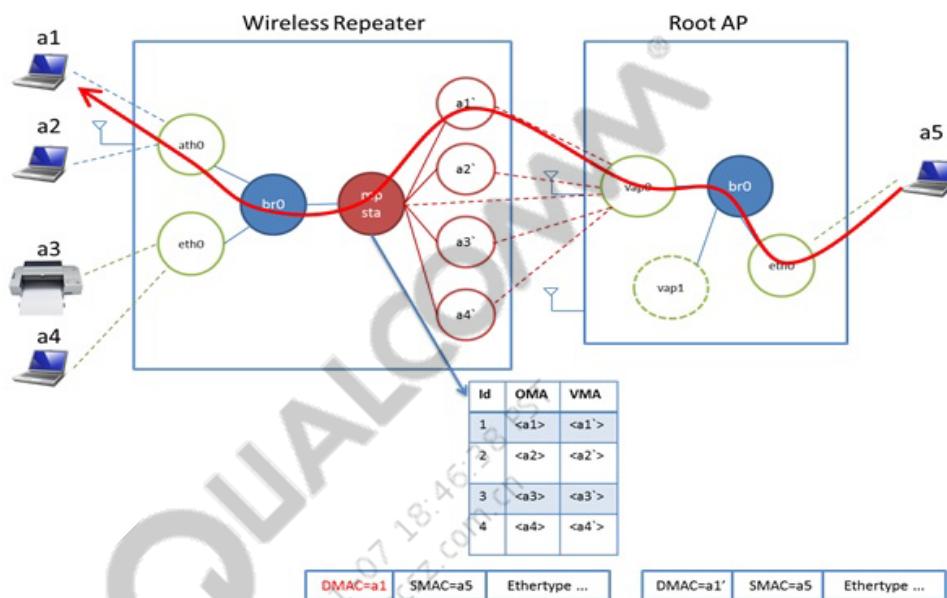
**NOTE** The “proxySTA” table is a static table maintained in the wireless driver. The entry for each client is added through configuration.

4. The VMA lookup is done only when the VAP Type == MP-STA.
5. If there is a lookup match, the Original MAC address (OMA) is translated to VMA and given to the respective VAP. For packets that have the MAC address embedded in them (for example, ARP, DHCP, ICMPv6-NDP), the driver should take care of translating the address at all the required places.
6. The packet is then transmitted by the respective MAC.

- The RootAP (vap0) receives the packet and hands it over to the Linux bridge. The bridge forwards it to the correct port, which is eth0 in this case. Station a5 receives the packet with a1' as the source mac address.

## Downlink

Figure 17-16 shows the data flow for the case of downlink – non-isolation.



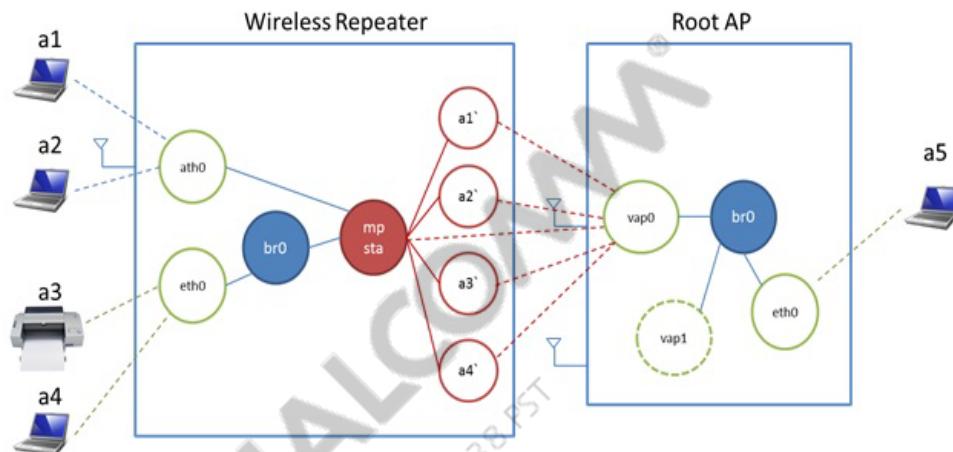
**Figure 17-16 Downlink – Non-isolation**

- Consider the case where an endpoint behind the RootAP “a5” intends to send a packet to the long-range client (proxySTA) “a1”. All the proxySTAs are known only by their virtual MAC addresses “a1’” outside the repeater AP. Hence, “a5” sends the packet with “a1’” as the destination address.
- When this packet is received by the RootAP, it is forwarded to the repeater AP.
- On the repeater AP, the packet is decrypted using a special hardware mode called the Azimuth mode, where the receiver address (and not the transmitter address) is used for key cache lookup.
- After decrypting the packet, the driver looks up the VMA in the proxySTA table and replaces a1' in the mac header with a1. Additionally, references to a1' in the payload (DHCP/ARP/NDP) are replaced with a1.
- The interface (netdev) is changed to MP-STA VAP and the packet is sent to the bridge ‘br0’.
- The bridge (which has already learnt about the network) forwards the packet to the proper destination (“ath0”).
- In the Qualcomm Technologies based IPQ806x platform, NSS replaces the bridge “br0” in the data path and forwards the packet to the proper destination after being handed over by the MP-STA ingress port.

The AP driver transmits the packet to the end station “a1” after proper encryption.

## Port Isolation feature

For the Q-WRAP port isolation feature, a few changes are required in the MP-STA driver (shown in [Figure 17-16](#)). For the isolation mode, all the traffic from wired and wireless stations should be forwarded to the RootAP without any local bridging/forwarding within the repeater. This is for cases where the firewall is implemented at the RootAP, and the RootAP wants to see all the packets from all the connected stations.



**Figure 17-17 Isolation mode**

[Figure 17-17](#) shows the modified topology for the isolation mode of Q-WRAP. The “ath0” VAP is removed from the bridge, since packets from a3/a4 clients destined for a1/a2 cannot be directly bridged and forwarded to the a1/a2 clients. These packets should be forwarded to the RootAP. The bridge on the RootAP forwards these packets back to the repeater (based on the firewall settings). The QSDK scripts should allow changing the topology dynamically for isolation and non-isolation cases.

In the downlink direction (RootAP to clients a1–a4), MP-STA needs to make a port forwarding decision – select one of the VAPs for wireless stations or send it to bridge (-> eth0) for wired stations. For this purpose, the VAP net device handle is stored in the proxySTA table. The MP-STA driver uses this handle to make the port forwarding decision.

| <b>Id</b> | <b>OMA</b> | <b>VMA</b> | <b>netdev</b>        |
|-----------|------------|------------|----------------------|
| 1         | <a1>       | <a1`>      | <ath0 netdev>        |
| 2         | <a2>       | <a2`>      | <ath0 netdev>        |
| 3         | <a3>       | <a3`>      | NULL for wired STA's |
| 4         | <a4>       | <a4`>      | NULL for wired STA's |

The port isolation feature also requires that the traffic from registered clients is allowed and traffic from unregistered clients is dropped by Q-WRAP. Hence, in this case, if the MP-STA driver does not find a client during the VMA lookup, it just drops the packet instead of forwarding.

### Multicast/Broadcast

- For Tx from the proxy STA VAP's to the RootAP, the regular Tx unicast processing data path can handle multicast/broadcast traffic, since the MAC table lookup is based on the source MAC.
- For Rx from RootAP to the proxy STA VAP, the MP-STA VAP receives multicast traffic. The Wi-Fi driver cannot do the MAC mangling for the destination MAC, but the device in the packet should be set to the MP-STA VAP while handing over to the network stack.

### Pseudocode

The pseudocode for changes in the wireless Rx and Tx functions (for both isolation and non-isolation cases) is given below.

#### Tx (hard\_start\_xmit)

The following handling of Q-Wrap is expected to be a part of hard\_start\_xmit of Wi-Fi at the beginning, as soon as the packet is received from the network stack:

- If `vap_type == MP_STA`
  - Extract the source mac address (SA) from the packet.
  - Use SA as a key to lookup the entry in proxySTA table. If the entry is found:
    - Change MAC address in the MAC header (OMA to VMA).
    - If protocol = ARP, DHCP, or IPv6NDP, replace the MAC address embedded in the payload.
    - Continue in the regular Tx path.
  - If the lookup failed, drop the packet.

#### Rx

The following handling of Q-Wrap is expected to be part of the Wi-Fi Rx, at the location where the packet is about to be handed over to the network stack:

- If vap\_type == WRAP\_AP and isolation ==1
  - Call MP-STA hard start.
- If vap\_type == PROXY\_STA,
  - Extract the destination MAC address (DA) from the packet.
  - Use DA as a key to lookup the entry in proxySTA table. If the entry is found:
    - Change MAC address in the MAC header (OMA to VMA).
    - If protocol = ARP, DHCP, or IPv6NDP, replace the MAC address embedded in the payload.
    - Change the interface (dev) to MP-STA dev.
    - Extract the client type from the proxySTA entry.
    - If (isolation\_mode == 1 && client\_type == wired) || (isolation\_mode == 0)
      - Do a netif\_rx() to hand over the packet to the bridge
    - Else
      - Call ath0 VAP hard\_start\_xmit.

#### 17.4.7.6 Configuration and automatic VAP setup (wrapd)

##### Overview

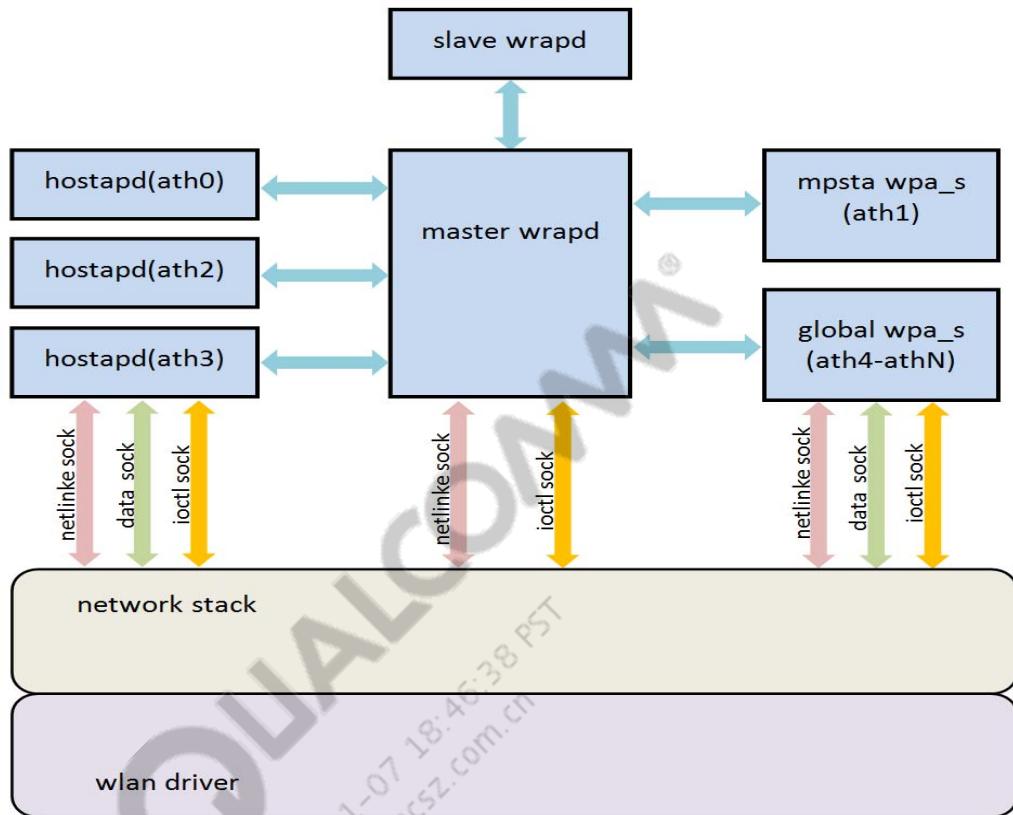
The wrapd contains two parts: master wrapd and slave wrapd.

The master wrapd runs as a user space demon, and handles all proxySTA configuration and setup. The master wrapd communicates with WLAN driver through the ioctl socket and netlink socket. At the same time, it communicates with hostapds and wpa\_supplicants through network sockets. Before master wrapd runs, all AP side hostapd processes and PSTA side wpa\_supplicant processes should be ready to access connections from control interfaces.

On the AP side, to support multiple AP VAPs with different security modes, master wrapd uses network socket to connect and attach to each hostapd for WPA/WPA2/WPS AP VAP, and listens on netlink socket for all OPEN/WEP AP VAPs.

On the PSTA side, the master wrapd also uses network sockets to build connections to both wpa\_supplicants: the single wpa\_supplicant for MPSTA VAP and the global wpa\_supplicant for PSTA VAPs. The MPSTA VAP is always available and the single wpa\_supplicant always tries to connect to the Root AP, and once the MPSTA gets connected/disconnected, it sends a CONNECT/DISCONNECT event to master wrapd through the network socket between them. Master wrapd listens on the event to detect Root AP link status.

Slave wrapd is a command line tool. It is used to add or remove an Ethernet client or list all the current Ethernet/wireless clients by sending a message through the wrapd ctrl interface. The master wrapd listens on the wrapd ctrl interface and manages these Ethernet clients.



**Figure 17-18 wrapd Block Diagram**

In Figure 17-18, there are three AP VAPs; each of them can be configured as WPA/WPA2 (including WPS) security modes.

For a wireless client, when it connects to any of the AP VAPs and 4-way handshake is done (for WPA/WPA2), the related hostapd will send a CONNECT message to the master wrapd; wrapd checks if the client is valid and adds it to an internal PSTA list, and if the link status between MPTA VAP and rootAP is UP, the master wrapd will create a PSTA VAP and add it into the global wpa\_supplicant. When a wireless client is disconnected, the related hostapd also sends a DISCONNECT message to master wrapd; master wrapd removes the related PSTA from the internal PSTA list, and if the link status between MPTA VAP and Root AP is UP, the PSTA will be removed from global wpa\_supplicant too.

For an Ethernet client, the command line provided by slave wrapd to add/remove/list Ethernet clients is used.

### 17.4.7.7 wrapd usage

#### master wrapd

Before master wrapd startup, make sure that all hostapds and wpa\_suplicants are ready to accept connections from the control interface.

```
wrapd -P /var/run/wrapd-global.pid
      -g /var/run/wrapdglobal
      -H /var/run/hostapd/global
      -w /var/run/wpa_supplicantglobal
      -b br-lan
      -i eth1
      -l 20
      -t 1
      -e 1
      -r 1 &
```

where,

/var/run/wrapdglobal - ctrl interface to add/remove wifi/ap/sta

/var/run/hostapd/global – ctrl interface to add/remove ap iface from global hostapd

/var/run/wpa\_supplicantglobal – ctrl interface to add/remove sta iface from global wpa supplicant

#### Slave wrapd

Ethernet clients can be added automatically/manually.

Command to add a client manually –

```
wrapd -S -g /var/run/wrapdglobal -A 00:11:22:33:44:55
```

Command to remove a client –

```
wrapd -S -g /var/run/wrapdglobal -R 00:11:22:33:44:55
```

To list all Ethernet and wireless clients extended through wifiX:

```
wrapd -S -g /var/run/wrapd-wifix -L
```

To list all Ethernet and wireless clients across all radios:

```
wrapd -S -g /var/run/wrapdglobal -L
```

### 17.4.7.8 DBDC Q-WRAP

The DBDC Q-WRAP uses two radios working on different bands (or non-overlapping channels on the same band) at the same time. It overcomes the multi-ACK/BA issue by physically isolating the root AP network and the repeater network into two collision domains. In this mode, one radio works in normal AP mode or Azimuth mode and the other one works in Azimuth mode. All the AP VAPs that are created on the radio work in normal mode, whereas all ProxySTA VAPs that are created on the radio work in the Azimuth mode. The performance is also doubled, as the wireless medium is separated into two collision domains. The DBDC Q-WRAP requires only an additional change for the configuration tool. The bridging change is identical with standard Q-WRAP.

The DBDC is currently supported with the following options:

- One radio has STA VAP (connect to Root AP) and the other radio has just AP VAP.
- One radio has AP VAP and STA VAP (connect to Root AP) and the other radio as just AP.
- Both the radios has AP VAP and STA VAP (connect to Root AP).

#### 17.4.7.9 Cascading multiple Q-WRAP devices

Cascading multiple Q-WRAP devices is easy and made straightforward by the Q-WRAP design. The only requirement is that additional unique MAC addresses (within the collision domain) need to be provided. The reason is that the previous 1-bit (LA) based MAC address translation cannot be used any more in the cascading case since it requires multiple times of translation. The mechanism to select locally unique MAC addresses is out of the scope of the Q-WRAP project.

In the cascading Q-WRAP configuration, each uplink Q-WRAP device is the “Root AP” of the current Q-WRAP device. The last Q-WRAP device accepts connections from both wireless clients and Ethernet clients. Note that an Ethernet client becomes a “wireless client” after it is cascaded with multiple Q-WRAP devices because Q-WRAP devices connect with each other only wirelessly.

#### 17.4.7.10 Proxy STA support on offload solutions

For solutions based on QCA988x/989x and QCA999x/998x, Proxy STA functionality is mostly accomplished in hardware.

With Proxy STA mode enabled, the host in addition to creating VAP entries will plumb the address of proxy devices in the Address Search Table (AST) using the WMI\_PDEV\_PARAM\_PROXY\_STA command.

With Proxy STA enabled, the hardware will perform RA look up (instead of TA lookup) on the AST entries when handling data frames.

##### **WMI command**

There is a single PDEV WMI command to set the device in proxy STA mode.

**WMI\_PDEV\_PARAM\_PROXY\_STA:** Place the hardware in proxy STA mode and do a RA look up for entries in the AST.

### 17.4.7.11 Automatic addition of Ethernet client to ProxySTA in QWRAP mode

#### Problem statement

In QWRAP mode, the AP has to detect the clients behind it in order to create proxy stations. For a wireless station the AP detects them when they are associated with the AP. But for a wired station, the AP does not detect the wired clients as they are outside the WLAN driver. The wired stations are connected to the bridge which is outside the WLAN driver.

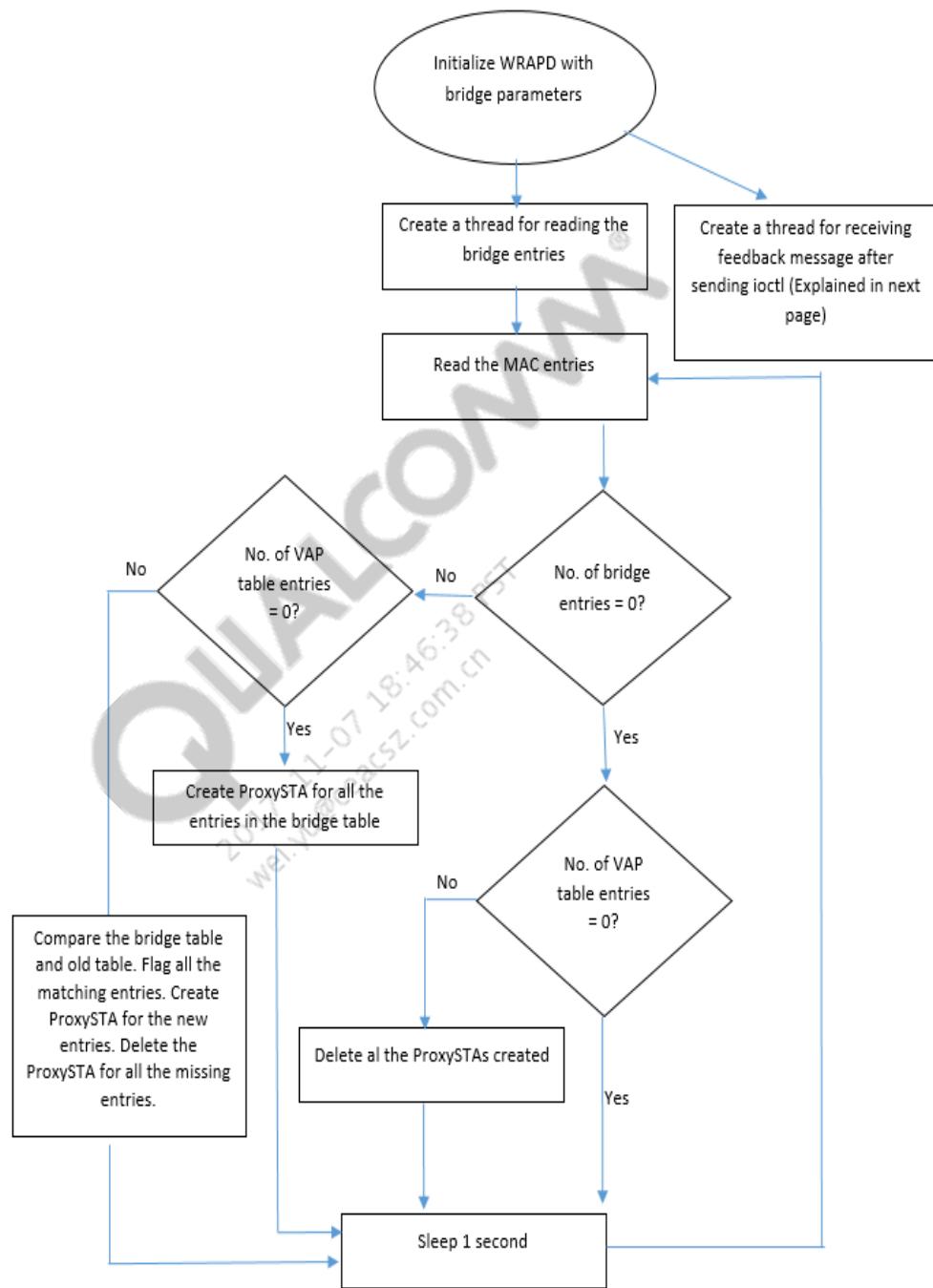
WRAPD app has the provision to add proxy stations manually. This requires human intervention each time a wired client is added. The AP needs to detect and create the proxy stations automatically without human intervention.

#### BRCTL

BRCTL is a software bridge that attaches the interfaces and forwards packets between them. The bridge intelligently learns the interface on which each mac address is connected with. It maintains a MAC address table and learning happens when frames are received. The bridge uses the MAC address table to forward the frames only to the appropriate interfaces. Whenever an entry is not present in the MAC address table, the frames are forwarded to all the interfaces (similar to broadcast).

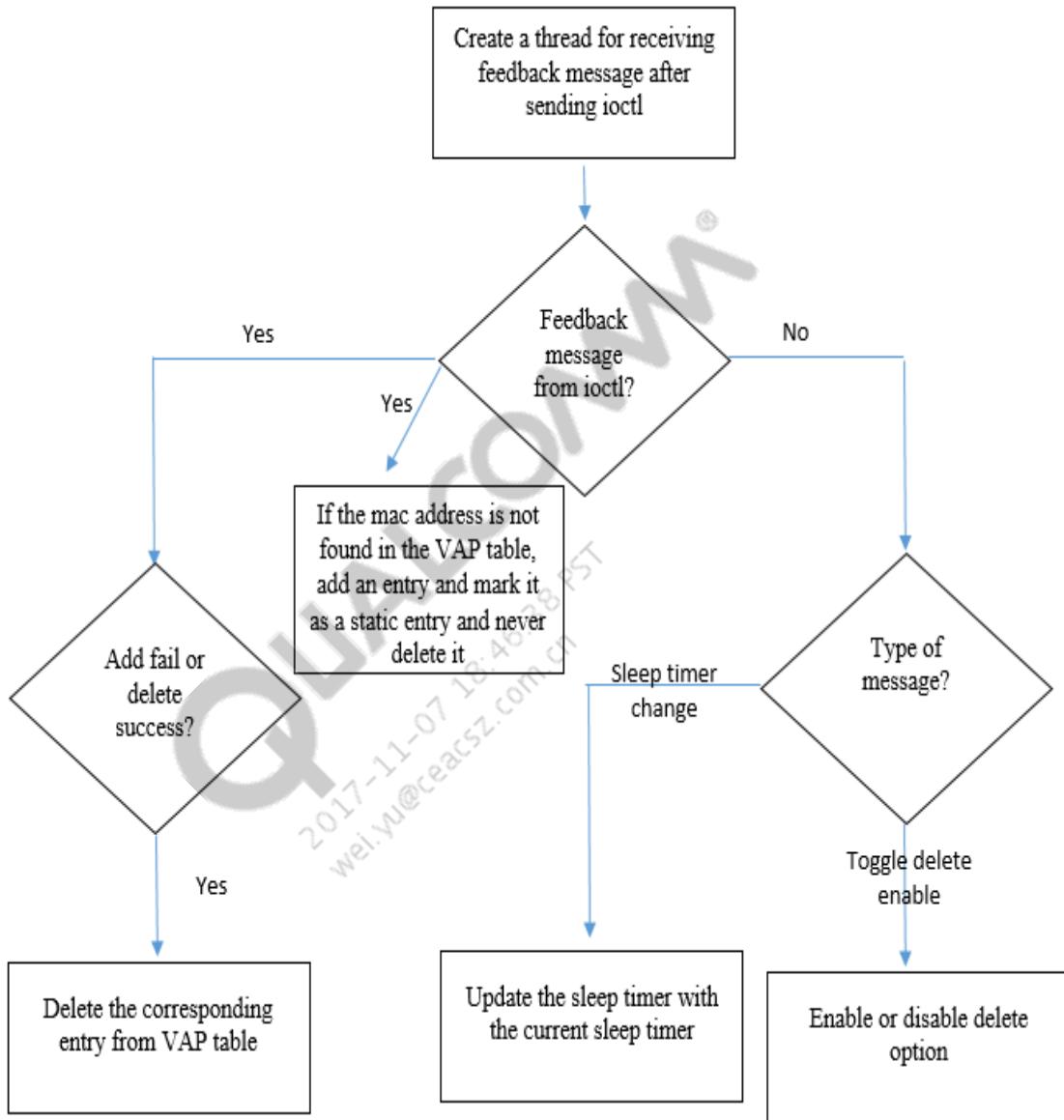
#### Solution

The bridge maintains a MAC table containing the MAC entries that can be reached from each interface. WRAPD app gets this list by reading the bridge entries in a fixed interval (ideally 1 second). The bridge entries are obtained from PROC entries or by passing an IOCTL (for old kernel). If any new entry is seen in the bridge MAC table, a proxySTA is created. If a proxySTA MAC address is not seen in the list, the proxySTA gets deleted (Deletion is optional and is configurable at runtime). Two threads are created for attaining the functionality. The main thread reads the bridge table, compares it with the existing Vap table and adds or deletes VAPs. The second thread is used for getting feedback message from the driver when vap creation is successful. This thread is also used to receive messages from another app for changing the parameters in runtime.



### Flowchart - Socket Feedback Thread

This thread receives the socket feedback message whenever there is an error in the VAP creation (ioctl error). The feedback message contains the mac address for which the ioctl was issued along with success or failure flag. This mac address is matched with the VAP table maintained and the entries for which the vap creation has failed are removed.



## Configuration

The WRAPD app must be configured with bridge parameters for polling information on bridge MAC table. These parameters are specified when WRAPD app is started in Master mode.

| Parameter | Description  |
|-----------|--|
| -b        | Used to enter the bridge name<br>For example, br-lan)  |
| -i        | Used to enter the LAN interface name.<br>Multiple interfaces can also be added by using this option multiple times<br>For example, eth1/eth0 |
| -t        | Used to specify the time in which the bridge mac table must be read.<br>Default time is set as 1 second                                      |
| -l        | Used to specify the Wi-Fi interface on which the Vap has to be created.<br>For example, wifi0  |
| -r        | Used to enable delete option   |

## 17.5 QWRAP support for Tri-Radio Standalone Repeater

| Feature   | IPQ4019.ILQ.4.0 |             |             |             | IPQ8064.ILQ.4.0 |             |            |             |             | QCA9531.ILQ.4.0 |             |             |             | QCA9558.ILQ.4.0 |             |             | QCA9563.ILQ.4.0 |             |             |             |   |
|---|-----------------|-------------|-------------|-------------|-----------------|-------------|------------|-------------|-------------|-----------------|-------------|-------------|-------------|-----------------|-------------|-------------|-----------------|-------------|-------------|-------------|---|
|   | IPQ<br>4019     | QCA<br>9886 | QCA<br>9984 | QCA<br>9889 | QCA<br>9984     | QCA<br>9980 | AR<br>9380 | QCA<br>9880 | QCA<br>9889 | QCA<br>9880     | QCA<br>9889 | QCA<br>9531 | QCA<br>9886 | QCA<br>9984     | QCA<br>9558 | QCA<br>9880 | QCA<br>9889     | QCA<br>9563 | QCA<br>9880 | QCA<br>9886 |   |
|   | 3.14            |             |             |             | 3.14            |             |            |             |             | 3.3.8           |             |             |             |                 |             |             |                 |             |             |             |   |
| QWRAP support for Tri-Radio Standalone Repeater | ✓               | ✓           | ✓           | x           | x               | x           | x          | x           | x           | x               | x           | x           | x           | x               | x           | x           | x               | x           | x           | x           | x |

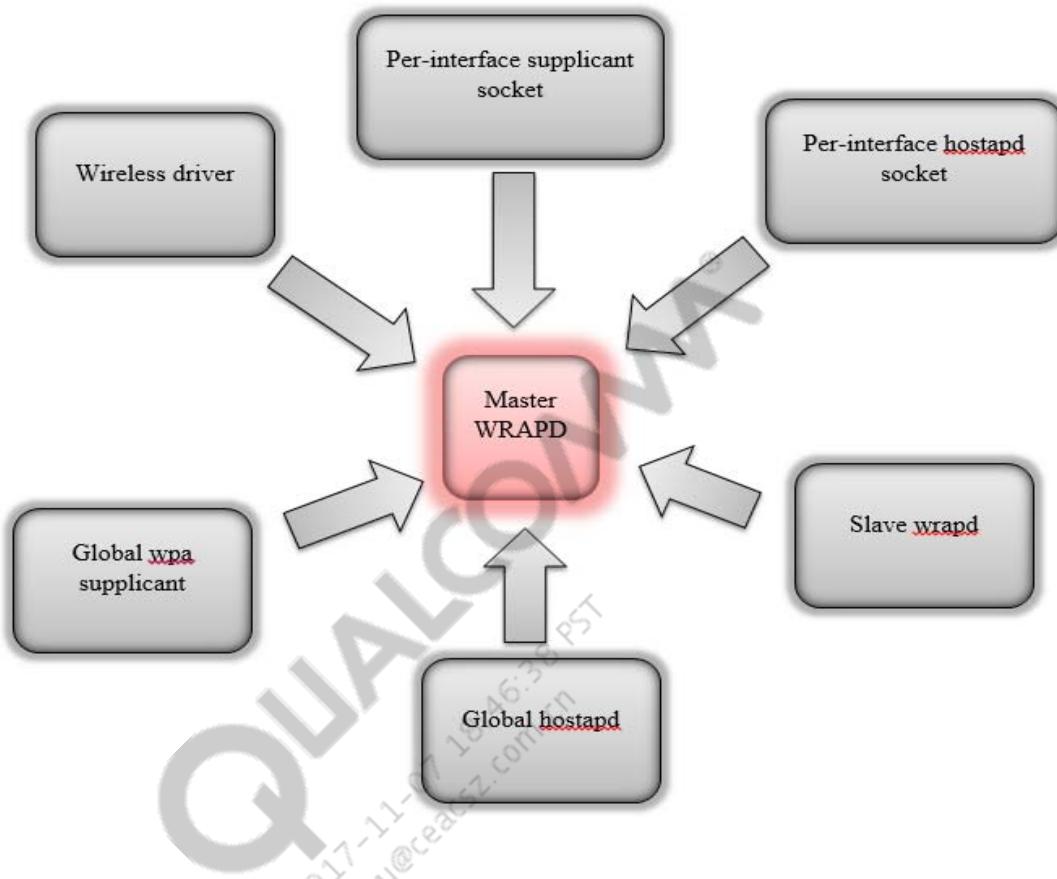
### 17.5.1 Feature requirement

Supports tri-radio standalone QWRAP along with all existing QWRAP features.

### 17.5.2 Design overview

Wrand code has been re-architected to support a master-slave model retaining features supported earlier.

Master wrand will process events below –



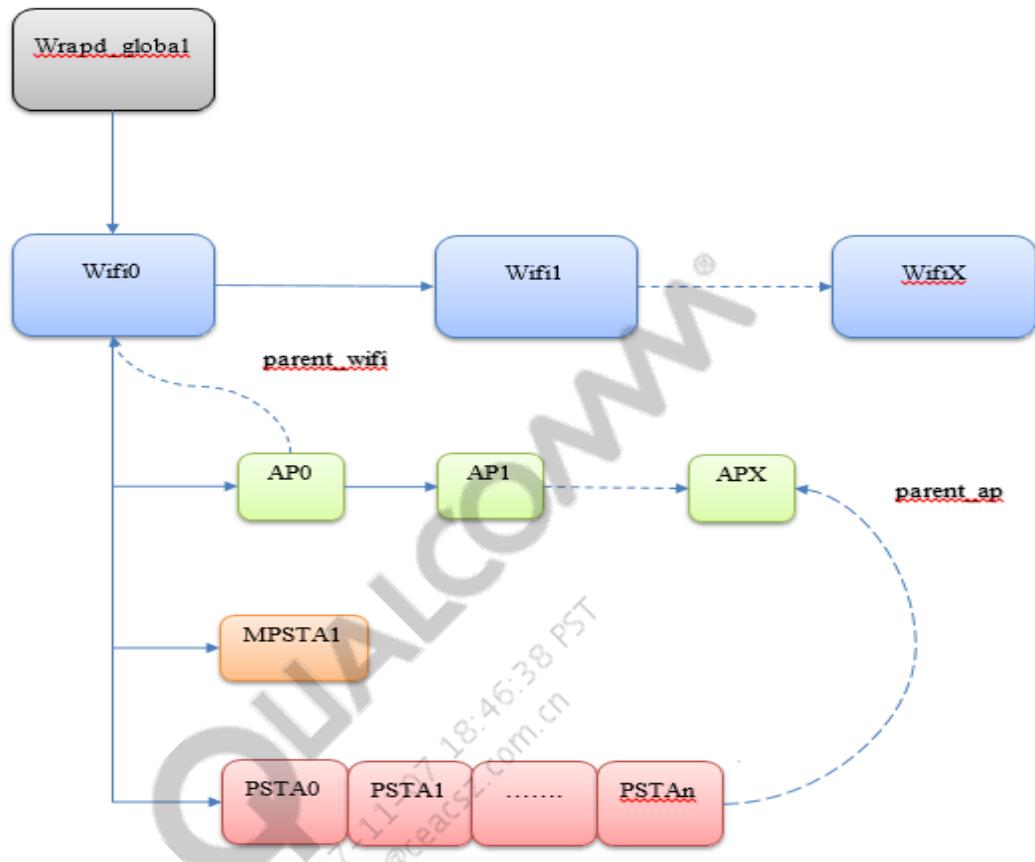
**Figure 17-19 WRAPD master-slave model**

1. Slave wrapd
  - a. Add or remove wifi interface
  - b. Add or remove ap interface
  - c. Add or remove mpsta interface
  - d. Add or remove wired clients manually
  - e. Add or remove wired clients automatically
  - f. List wireless clients on each qwrap radio
2. Global hostapd
  - a. Master wrapd issues command to add or remove AP through global hostapd process
3. Global wpa supplicant
  - a. Master wrapd issues commands to add or remove supplicant interface through global wpa supplicant process
4. Per interface wpa\_supplicant socket
  - a. Wrapd receives CTRL-EVENT-DISCONNECTED/ CTRL-EVENT-CONNECTED when mpsta disconnects/connects to Root AP

5. Per interface hostapd socket
  - a. Wrapd received AP-STA-CONNECTED/ AP-STA-DISCONNECTED events for station connection/disconnected to QWRAP AP.
6. Wireless driver
  - a. Wrapd queries driver through IOCTL –
    - i. IEEE80211\_IOCTL\_KICKMAC
    - ii. IEEE80211\_IOCTL\_GETPARAM
    - iii. EXTENDED\_SUBIOCTL\_OL\_RESERVE\_PROXY\_MACADDR
    - iv. EXTENDED\_SUBIOCTL\_OL\_GET\_PROXY\_NOACK\_WAR
    - v. EXTENDED\_SUBIOCTL\_GET\_MPSTA\_MAC\_ADDR
    - vi. EXTENDED\_SUBIOCTL\_GET\_FORCE\_CLIENT\_MCAST
    - vii. EXTENDED\_SUBIOCTL\_GET\_MAX\_PRIORITY\_RADIO

### 17.5.3 Data structure

- wrapd\_global is a global structure that has the list of wifi devices which are configured in azimuth mode
- Each wifi device has a list of AP devices(hapd\_list). For a station connecting to any of the APs in this list, corresponding proxy station is created on this wifi device
- Each wifi device has its corresponding MPSTA, the connection state of MPSTA decides the connection state of other proxy stations on that radio
- ‘parent\_wifi’ of an AP is the wifi which the user originally configures as extender of the AP
- ‘parent\_ap’ of a PSTA is the AP to which the real client is connected



**Figure 17-20 WRAPD data structure design**

#### 17.5.4 Route traffic always through primary radio

When there are multiple paths for the Repeater to reach the Root, traffic can always be routed through the primary radio even if other UPLINKS are active.

With this configuration, all uplink traffic between Repeater and Root always uses only one of the radios. We create a copy of the proxy stations created on the primary radio, on all other QWRAP radios. This helps in smooth transition during failsafe.

##### Implementation

- All stations are registered to WRAPD before APs
- For a wireless client connecting to an AP, proxy stations are created on all available qwrap radios
- For a wired client, added either manually (or) automatically proxy stations are created on all available qwrap radios
- This feature has a limitation that the entire system can support only the maximum number of clients supported by a single qwrap radio

### 17.5.5 Failsafe

Given that, the Repeater has multiple paths to reach the Root, if an uplink fails, we provide a failsafe mechanism, where the max priority active uplink connection is used to route the packets to the Root.

- Driver computes maximum priority active radio based on user's primary radio setting and active uplink available at any point of time
- WRAPD retrieves the max priority active radio through IOCTL
- Switching happens for following 3 cases –
  - MPSTA disconnected
    - Move all child PSTAs/APs to max priority active radio
  - MPSTA connected and it is the max priority active radio
    - Move wired, child and fostered PSTAs/APs to max priority active radio
  - MPSTA connected and it is not the max priority active radio
    - Move all child PSTAs/APs to the parent qwrap radio

### 17.5.5.1 Failsafe when MPSTA is disconnected

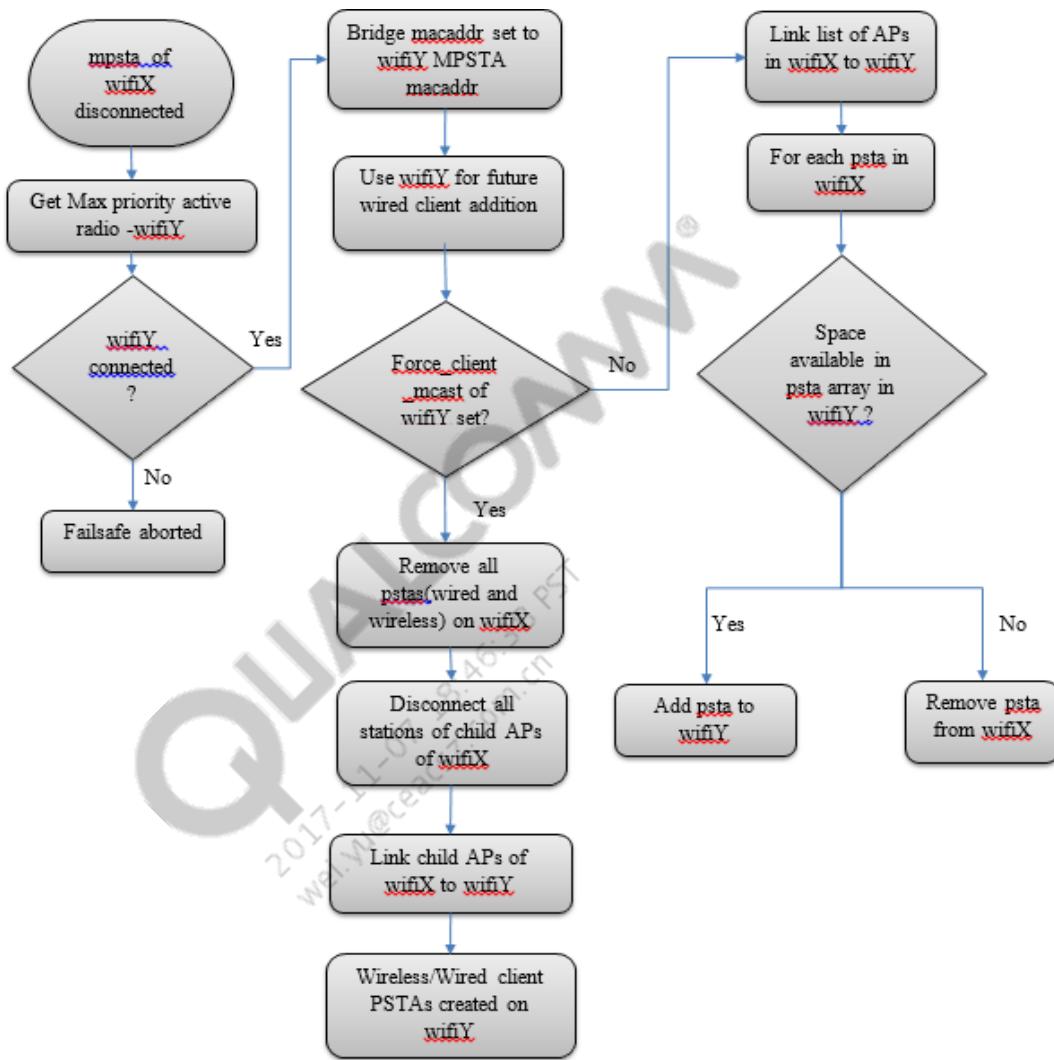


Figure 17-21 Scenario - MPSTA disconnected

### 17.5.5.2 Failsafe when MPSTA is connected

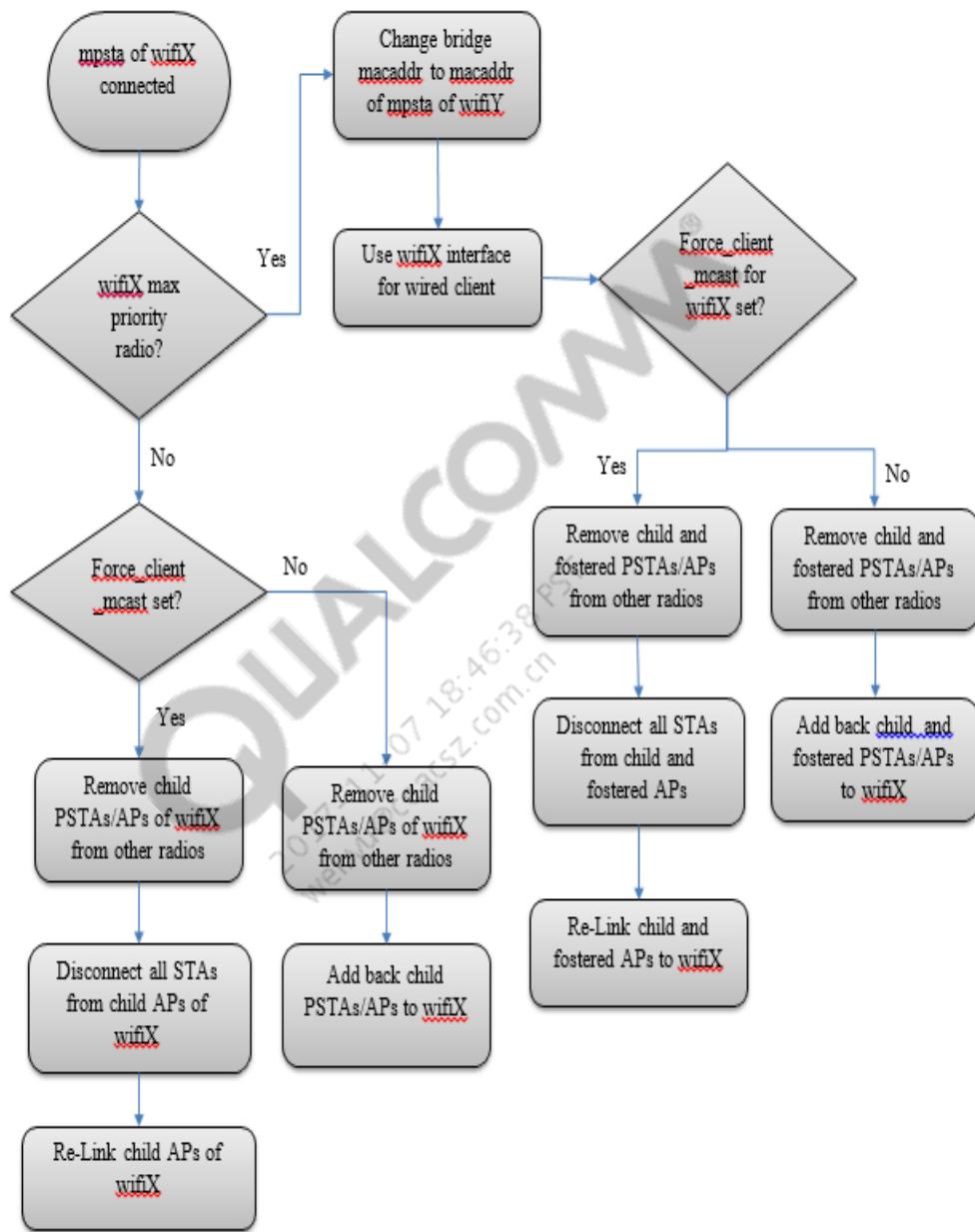


Figure 17-22 Scenario - MPSTA connected

### 17.5.6 Wired client addition

- Wired clients can be added either manually or automatically
- Wired clients are always extended through max priority active radio
- WRAPD keeps updating the max priority active radio for any MPSTA connection or disconnection event

**NOTE** WAL\_DBGID\_DEV\_RESET error is noticed on repeater APs. These errors occur with 2 GHz and 5 GHz configured with QWRAP repeater AP and UDP bidirectional traffic sent. This repeater AP associates with root AP in 5 GHz. This is an expected behavior in a noisy environment.

**NOTE** WAL\_DBGID\_DEV\_TX\_TIMEOUT and WAL\_DBGID\_DEV\_RX\_TIMEOUT errors are observed. These errors occur with 2 GHz and 5 GHz configured with QWRAP root AP and UDP bidirectional traffic sent. These error messages occur on different platforms, such as IPQ4019 and QCA9980, do not affect the functionality and throughput.

## 17.6 DBDC Repeater

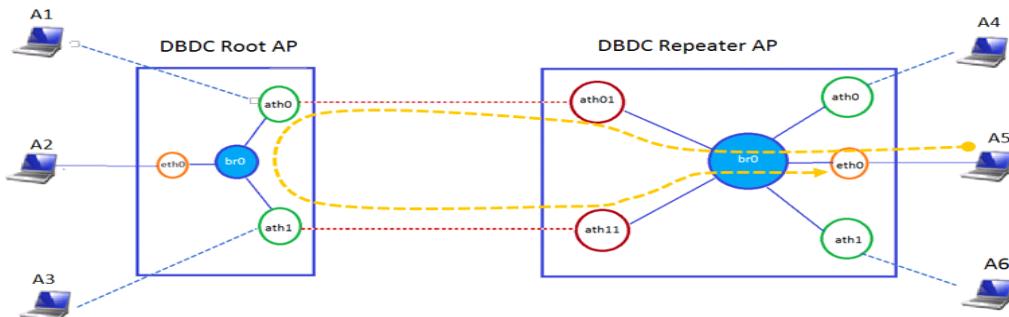
AP works on repeater mode on both 2.4GHz and 5GHz radios concurrently. Each radio can create two VAPs. One VAP acts as the AP and the other VAP acts as the client. The AP VAP allows clients to associate to itself. The STA-VAP associates with the Root AP and bridges the wireless and wired frames to the Root AP. DBDC Repeater should be configured on WDS / EXTAP/ QWRAP mode on both radios.

### 17.6.1 Challenges for DBDC Repeater

#### 17.6.1.1 Looping of Multicast and Broadcast packets

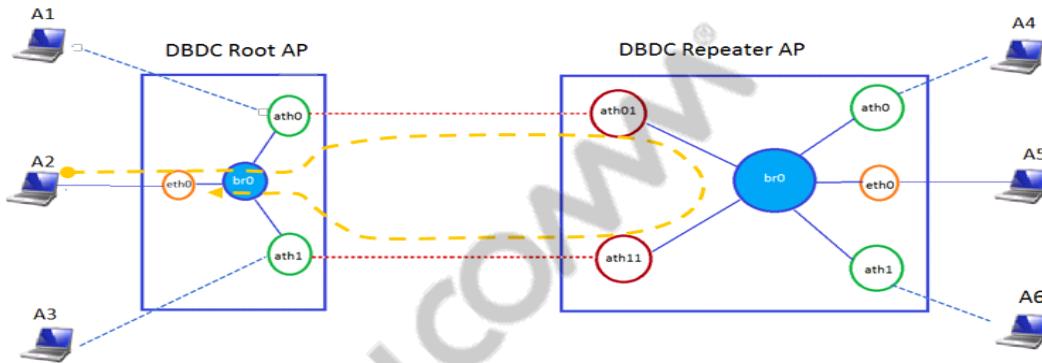
Multicast and Broadcast packets will get looped when both radios of the DBDC Repeater are connected to DBDC RootAP. On DBDC Repeater, AP VAPs and STA VAPs of both radios are connected on common bridge. Repeater Bridge forwards any multicast / broadcast packet received on one VAP to all other VAPs. Similarly on DBDC RootAP, AP VAPs of both radios are connected on common Bridge.

As shown in below diagram, multicast / broadcast packet that is sent by Ethernet client A5 is received on Repeater Bridge and then Bridge forwards that packet to all VAPs. DBDC Repeater sends that packet to RootAP through both ath01 and ath11 interface. Packet that is sent on one STA VAP will be received on other STA VAP.



ath0 – AP VAP of 2.4 GHz radio.  
 ath1 – AP VAP of 5 GHz radio.  
 ath01 – STA VAP of 2.4 GHz radio.  
 ath11 – STA VAP of 5 GHz radio.

In another case, multicast / broadcast packet that is sent by client A2 will get looped because both ath01 and ath11 of DBDC Repeater are connected on common bridge.



### 17.6.1.2 Achieving Load balance

Traffic from client connected with DBDC Repeater will reach DBDC RootAP either through 2.4 GHz STA VAP or 5 GHz STA VAP based on Repeater bridge learning. For example, If ARP request (broadcast packet) is sent from DBDC Root AP on both AP VAPs, DBDC Repeater will get that packet on both the STA VAPs. Repeater Bridge will update its forwarding table based on latest received frame. ARP reply (unicast packet) from client connected with DBDC Repeater can be sent on either on 2.4 GHz or 5 GHz STA VAP.

In order to achieve the load balance, below two conditions should be met,

- Traffic from clients connected with 2.4 GHz radio should use only 2.4 GHz STA VAP to reach DBDC Root AP.
- Traffic from clients connected with 5 GHz radio should use only 5 GHz STA VAP to reach DBDC Root AP.

### 17.6.1.3 Ethernet client traffic on desired radio's STA VAP

Based on user configuration, Ethernet client traffic from DBDC Repeater should be send on desired radio's STA VAP to reach DBDC RootAP. User can able to change this configuration dynamically and allow Ethernet client traffic to be sent on either 2.4 GHz STA VAP or 5 GHz STA VAP.

### 17.6.1.4 Auto Detection of DBDC RootAP connection

When DBDC Repeater is connected with two different RootAPs, looping of multicast / broadcast packets won't occur and based on Repeater bridge learning packet will be forwarded to either 2.4

GHz or 5 GHz STA VAP to reach destination associated with anyone of the RootAPs. DBDC Repeater should able to identify the connection with DBDC RootAP without any user inputs.

## 17.6.2 Design for DBDC Repeater

One of the two radios of the DBDC Repeater is configured as a primary radio and the other radio is configured as a secondary radio. By default, 2.4 GHz radio would be set as the primary radio and 5 GHz radio would be set as secondary radio. DBDC Repeater can dynamically configure either radio as the primary radio based on user inputs. The primary radio is set as the radio to provide the root AP link for Ethernet clients of DBDC Repeater. DBDC Repeater would send Ethernet client traffic through STA VAP associated with the primary radio to reach DBDC RootAP.

### 17.6.2.1 Detection of packet loop

DBDC Repeater would detect a packet loop by checking if a received multicast / broadcast packet is sent from the DBDC repeater AP. DBDC Repeater would check source address of received multicast / broadcast packet to determine whether it matches with the MAC address of any STA VAPs on DBDC Repeater or anyone of the client associated with DBDC Repeater. Common table is used to store radio structure pointers of both 2.4 GHz and 5 GHz radio. This common connection table is needed to identify the client associated with each radio.

Spanning Tree Protocol (STP) cannot be used to detect the packet loop because it disables the port which is causing the packet loop. Load balancing cannot be achieved if STP disables any port. Detect packet loop by sending L2UF frame (Layer 2 update frame) from WLAN driver. The layer-2 update frame is an 802.2 LLC frame. Generally, this frame is sent from AP with source address as MAC address of the newly associated station and destination address as broadcast address. Upon the reception of this frame, any layer-2 devices (for example, bridges, switches, and other APs) update their forwarding tables with the correct port to reach the new location of the station.

After STA VAP of DBDC Repeater associates with DBDC Root AP, check if both radio's STA VAP are connected, if so send L2UF frame from both STA VAPs. If L2UF frame received on any STA VAP with MAC address of another radio's STA VAP, packet loop is detected and this loop is because of connection with DBDC Root AP.

In this method, received L2UF frame / MCAST frame is used to determine whether a packet loop exists or not.

- In EXTAP mode, during multicast packet receive, source MAC address of multicast packet will be checked with STA VAP MAC address of other radio. If it matches, then packet loop is detected.
- In WDS mode, during multicast packet receive, source MAC address of multicast packet will be checked with MAC address of all clients connected with other radio. If it matches, then packet loop is detected.
- In QWRAP mode, during multicast packet receive, source MAC address of multicast packet will be checked with all Proxy-STA addresses present in other radio. If it matches, then packet loop is detected.

This above loop detection check will be done only when VAP is in STA mode, number of connected STA VAPs is greater than one and loop is not detected before.

When any one of the STA VAP disconnects, then loop detection flag will be reset. So that, this loop detection logic will kick in every time after second STAVAP connection.

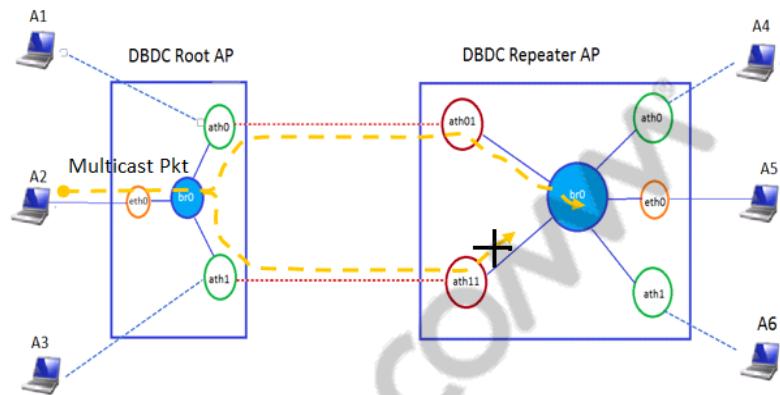
**NOTE** With root AP and QWRAP AP brought up in DBDC mode, and QWRAP AP associated with root AP, if the QWRAP AP is powered on and off, association takes 30 seconds longer in the QCA\_Networking\_2016.SPF.4.0 release than it takes in QCA\_Networking\_2016.SPF.3.0 after the QWRAP AP is powered on. This is an expected behavior.

## 17.7 Special Packet handling for Loop avoidance and Load balancing

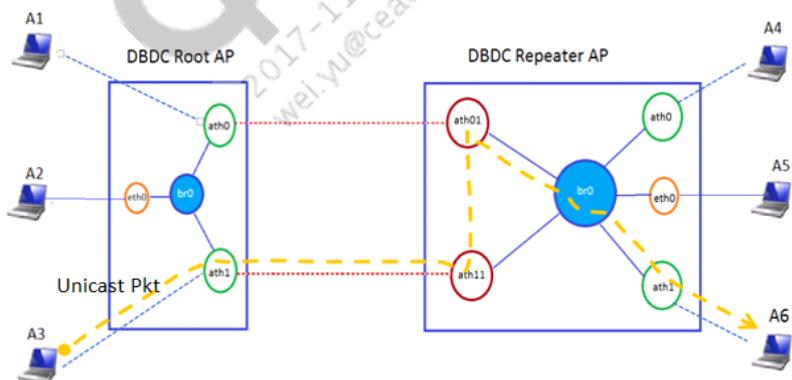
| Feature   | IPQ4019.ILQ.4.0 |             |             |             | IPQ8064.ILQ.4.0 |             |            |             |             | QCA9531.ILQ.4.0 |             |             |             | QCA9558.ILQ.4.0 |             |             | QCA9563.ILQ.4.0 |             |             |             |   |
|---|-----------------|-------------|-------------|-------------|-----------------|-------------|------------|-------------|-------------|-----------------|-------------|-------------|-------------|-----------------|-------------|-------------|-----------------|-------------|-------------|-------------|---|
|   | IPQ<br>4019     | QCA<br>9886 | QCA<br>9984 | QCA<br>9889 | QCA<br>9984     | QCA<br>9980 | AR<br>9380 | QCA<br>9880 | QCA<br>9889 | QCA<br>9880     | QCA<br>9889 | QCA<br>9531 | QCA<br>9886 | QCA<br>9984     | QCA<br>9558 | QCA<br>9880 | QCA<br>9889     | QCA<br>9563 | QCA<br>9880 | QCA<br>9886 |   |
|   | 3.14            |             |             |             | 3.14            |             |            |             |             | 3.3.8           |             |             |             |                 |             |             |                 |             |             |             |   |
| TX data flow of DBDC Repeater based on different configurations | ✓               | ✓           | ✓           | ✓           | ✓               | ✓           | ✓          | ✓           | ✓           | x               | x           | x           | x           | x               | x           | x           | x               | x           | x           | x           | x |
| RX data flow of DBDC Repeater based on different configurations | ✓               | ✓           | ✓           | ✓           | ✓               | ✓           | ✓          | ✓           | ✓           | x               | x           | x           | x           | x               | x           | x           | x               | x           | x           | x           | x |

When loop is detected, received packet on DBDC Repeater from DBDC RootAP should be handled as mentioned below,

- The secondary radio's STA VAP will ignore the multicast/broadcast packets received from DBDC RootAP and let them be processed only through the primary radio STA VAP interface. This helps to avoid the looping of multicast packets.



- When unicast packet from DBDC RootAP is received on secondary radio's STA VAP, modify packet header to indicate Repeater Bridge that the unicast packet comes from primary radio's STA-VAP Hence, the Repeater Bridge learns all MAC address associated with DBDC RootAP as reachable through primary radio's STA VAP only.

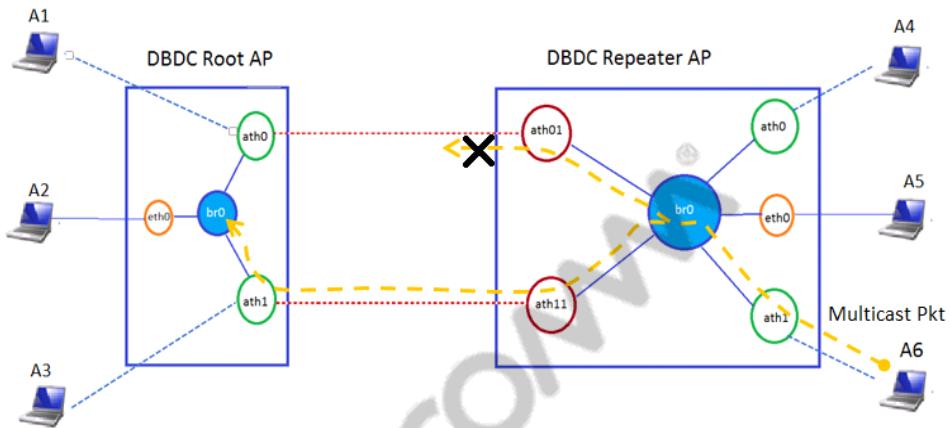


- For the multicast / broadcast / unicast packets received from DBDC RootAP on the primary radio's STA VAP of the repeater, there is no change other than regular packet processing as is already done.

When loop is detected, packet that needs to be transmitted from DBDC Repeater to DBDC RootAP should be handled as mentioned below,

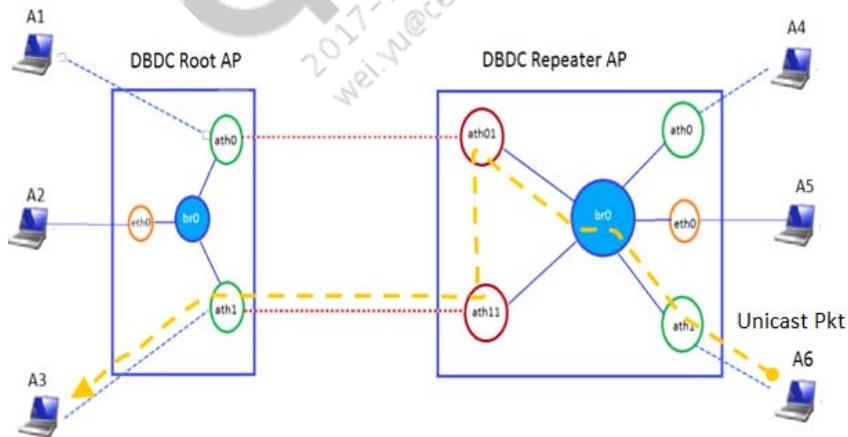
- When the bridge of the repeater gives multicast packet to the primary radio's STA VAP, the packet will be discarded if the source MAC address is matching with any of the client associated with secondary radio. This in essence means that the multicast packet is transmitted if the originator of this packet belongs to the primary radio's AP VAP or the Ethernet network.

- When the bridge of the repeater gives multicast packet to the secondary radio's STA VAP, send the packet only if the source MAC address is matching with any of client associated with secondary radio. This in essence means that all packets originating from clients of primary radio and Ethernet network will be dropped.



- Repeater Bridge will always handover unicast packets only to the primary radio's STA VAP for further transmission to DBDC RootAP. When such unicast packet is received from bridge, the default radio's STA VAP does the following,

1. Hand over that packet to secondary radio's STA VAP if the packet source MAC address is matching with any of the client associated with secondary radio.



- Transmit the packet as if the originator of the packet belongs to the primary radio's AP VAP or the Ethernet network.

When the STA VAP of any of the radio loses connectivity with DBDC RootAP, trigger that to indicate that the loop is no longer present so that the above mentioned implementation logic is turned off and regular packet flow processing is used. The following table informations are common for all repeater modes (WDS, EXTAP and QWRAP).

Table 17-2 shows the TX data flow of DBDC Repeater based on different configurations.

**Table 17-2 TX data flow of DBDC Repeater based on different configurations**

| Mcast/<br>uCast | Always_<br>primary | Loop<br>detected | TX on primary sta vap  | TX on non-primary sta vap   |
|-----------------|--------------------|------------------|--|---|
| Mcast           | 0                  | 0                | -  | -   |
| Mcast           | 0                  | 1                | Send all other remaining mcast traffic (primary radio clients, Ethernet, bridge and other clients mcast traffic) | Send only corresponding radio mcast traffic                             |
| Mcast           | 1                  | 0                | Send all mcast traffic received from bridge  | Drop mcast traffic, except EAPOL  |
| Mcast           | 1                  | 1                | Send all mcast traffic received from bridge  | Drop mcast traffic, except EAPOL  |
| Ucast           | 0                  | 0                | -  | -   |
| Ucast           | 0                  | 1                | Send all other remaining ucast traffic (primary radio clients, Ethernet, bridge and other clients ucast traffic) | Send only corresponding radio ucast traffic                             |
| Ucast           | 1                  | 0                | Send all ucast traffic received from bridge  | Send ucast pkts through primary sta vap, which are received from bridge |
| Ucast           | 1                  | 1                | Send all ucast traffic received from bridge  | Send ucast pkts through primary sta vap, which are received from bridge |

Table 17-3 shows the RX data flow of DBDC Repeater based on different configurations.

**Table 17-3 RX data flow of DBDC Repeater based on different configurations**

| Mcast/<br>uCast | Always_<br>primary | Loop<br>detected | RX on primary sta vap     | RX on non-primary sta vap  |
|-----------------|--------------------|------------------|---------------------------|--|
| Mcast           | 0                  | 0                | -                         | -  |
| Mcast           | 0                  | 1                | Pass mcast pkts to bridge | Drop mcast pkts  |
| Mcast           | 1                  | 0                | Pass mcast pkts to bridge | Drop mcast pkts  |
| Mcast           | 1                  | 1                | Pass mcast pkts to bridge | Drop mcast pkts  |
| Ucast           | 0                  | 0                | -                         | -  |
| Ucast           | 0                  | 1                | Pass ucast pkts to bridge | Pass ucast pkt through bridge by changing skb dev as primary sta vap. So that bridge learns, that this pkt src mac is reachable through primary sta vap. |
| Ucast           | 1                  | 0                | Pass ucast pkts to bridge | Pass ucast pkt through bridge by changing skb dev as primary sta vap. So that bridge learns, that this pkt src mac is reachable through primary sta vap. |
| Ucast           | 1                  | 1                | Pass ucast pkts to bridge | Pass ucast pkt through bridge by changing skb dev as primary sta vap. So that bridge learns, that this pkt src mac is reachable through primary sta vap. |

### 17.7.1 Implementation of Tx and Rx data flows for DBDC repeater

New common data structure ‘global\_ic\_list’ is used to store UMAC radio device structure pointers (struct ieee80211com \*ic). This data structure is common for both radios. Using this structure, each radio can able to check the MAC address of client associated with other radio.

Received packets are processed on dbdc\_rx\_process function, which is called from osif\_deliver\_data function. Packets that need to be transmitted from Repeater are processed on dbdc\_tx\_process function, which is called from osif\_hardstart function. DBDC Repeater process are done in both direct attach and offload path.

DBDC Repeater changes are included under **DBDC\_REPEATERSUPPORT** compiler flag.

- If the platform is tri-radio and has NSS wifi offload support, then for TBTC Repeater feature to work, NSS wifi offload has to be disabled through UCI command

User need to provide the below user level configuration to disable NSS wifi offload mode.

```
uci set wireless.qcawifi=qcawifi
uci set wireless.qcawifi.nss_wifi_olcfg=0
uci commit
```

- By default, wifi0 will be set as primary radio. On tri-radio board, if wifi0 is disabled and if wifi1 and wifi2 are configured in DBDC Repeater mode, then user needs to explicitly set wifi1 or wifi2 as primary radio.
- Dynamically changing `alwaysprimary` configuration during runtime is not supported. The `alwaysprimary` configuration has to be set only through UCI.

### 17.7.2 Configuration of Tx and Rx data flows for DBDC repeater

Set `alwaysprimary` flag as 0, when user wants DBDC Repeater to send wireless client traffic through corresponding radio's STA VAP and ethernet client traffic through primary radio's STA VAP.

Set `alwaysprimary` flag as 1, when user wants DBDC Repeater to send all wireless and wired clients traffic through primary radio's STA VAP.

By default, `alwaysprimary` flag will be set as 0. `alwaysprimary` flag is a system flag.

To set wifi0 as primary radio, please use below commands.

```
uci set wireless.@wifi-device[0].primaryradio=1
uci commit wireless
```

To set wifi1 as primary radio, please use below commands.

```
uci set wireless.@wifi-device[1].primaryradio=1
uci commit wireless
```

To know which radio is configured as primary radio, use below command:

```
iwpriv wifiX getprimaryradio
```

To set alwaysprimary flag, use below commands

```
uci set wireless.qcawifi=qcawifi  
uci set wireless.qcawifi.alwaysprimary=1  
uci commit wireless
```

When DBDC Repeater is connected with two different RootAPs and if user wants to disable special handling of packets on DBDC Repeater, following command can be used. By default, this will be enabled on DBDC Repeater. dbdc\_enable flag is a system flag.

```
iwpriv wifiX dbdc_enable 0
```

To enable special handling of packets on DBDC Repeater, use the following command:

```
iwpriv wifiX dbdc_enable 1
```

To get current configuration, use the following command:

```
iwpriv wifiX get_dbdc_enable
```

QUALCOMM  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

## 17.8 Interface Manager (iface\_mgr) application

| Feature                                   | IPQ4019.ILQ.4.0 |             |             |             | IPQ8064.ILQ.4.0 |             |            |             |             | QCA9531.ILQ.4.0 |             |             |             | QCA9558.ILQ.4.0 |             |             | QCA9563.ILQ.4.0 |             |             |             |
|---|-----------------|-------------|-------------|-------------|-----------------|-------------|------------|-------------|-------------|-----------------|-------------|-------------|-------------|-----------------|-------------|-------------|-----------------|-------------|-------------|-------------|
|   | IPQ<br>4019     | QCA<br>9886 | QCA<br>9984 | QCA<br>9889 | QCA<br>9984     | QCA<br>9980 | AR<br>9380 | QCA<br>9880 | QCA<br>9889 | QCA<br>9880     | QCA<br>9889 | QCA<br>9531 | QCA<br>9886 | QCA<br>9984     | QCA<br>9558 | QCA<br>9880 | QCA<br>9889     | QCA<br>9563 | QCA<br>9880 | QCA<br>9886 |
|   | 3.14            |             |             |             | 3.14            |             |            |             |             | 3.3.8           |             |             |             |                 |             |             |                 |             |             |             |
| Interface Manager (iface_mgr) application | ✓               | ✓           | ✓           | ✓           | ✓               | ✓           | ✓          | ✓           | ✓           | x               | x           | x           | x           | x               | x           | x           | x               | x           | x           | x           |

This application acts as an interface manager to bring down/up AP VAPs of any radio based on STA VAP connection. Each vap has to be added to anyone group, by default all AP VAPs and STA VAPs will get added to group 0.

This application establishes network socket connection with wpa\_supplicant in order to get CONNECT/DISCONNECT message during sta vap connection or disconnection with Root AP. When connection to wpa\_supplicant will fail, a retry for the connection will happen till a connection has been established with wpa\_supplicant; and once the connection is established, a ping message will be sent in every five seconds to check the liveness.

This application will be started as part of post qcawifi script and it will be restarted every time during wifi down and wifi up. This application can be configured on following modes:

- global\_wds mode
- fast\_lane mode

**NOTE** If lbd.config.Enable is set as 0, lbd is not started in the interface manager (iface-mgr) application.

### 17.8.1 global\_wds mode

Post qcawifi script will create configuration file based on provided UCI options at “/var/run/iface\_mgr.conf” path. Configuration file will be created on below format.

```
mode=0          //0-global_wds
timeout=50    // timeout to bring down AP VAPs, when STA VAP goes down.

group=0 ap_vap=ath0
group=0 sta_vap=ath01
group=0 ap_vap=ath1
```

By default, all AP VAPs and STA VAPs are added on group 0.

To change the group for any VAP, use the following UCI for corresponding VAP:

```
uci set wireless.@wifi-iface[0].group=2
uci commit
```

When application starts, it reads inputs from this configuration file.

## Timeouts

The following two timeouts are provided by user.

- Stavap\_disconnection\_timeout—Period of time after which the STA VAP establishes a reconnection with RootAP.
- RootAP\_reconfiguration\_timeout—Period of time after which the RootAP can be reconfigured.

A maximum of two timeouts are added in the configuration file. The application retrieves the timeout value from the configuration file.

The default Stavap\_disconnection\_timeout is 10 secs and the default RootAP\_reconfiguration\_timeout is 60 secs. Later, when user changes timeout value through iwpriv command, the application uses the updated timeout value from driver.

The following UCI and iwpriv commands are available to update timeout value on driver and configuration file:

```
uci set wireless.qcawifi.discon_time='20'
uci set wireless.qcawifi.reconfig_time='45'

iwpriv wifix discon_time 20

iwpriv wifix gdiscon_time

iwpriv wifix reconfig_time 45

iwpriv wifix greconfig_time
```

When last STA VAP connection goes down on any group, bring down all AP VAPs belong to that group after specified timeout. When first STA VAP connection comes up on any group, bring up all AP VAPs belong to that group.

The application handles the bringing down/up events of AP VAPs based on STA VAP connection. Design changes are implemented in the driver to not bring up /down AP VAPs when the STA VAP of the corresponding radio connects to or disconnects from the root AP.

This application will run when global\_wds flag is set through UCI.

```
uci set wireless.qcawifi=qcawifi
uci set wireless.qcawifi.global_wds=1
```

WDS repeater operates on Enhanced Independent repeater (EIR) mode, when this feature is enabled.

## 17.8.2 fast\_lane mode

Fast lane is created between one STA VAP of one 5 GHz radio and one AP VAP of another 5 GHz radio in a tri-radio board. To enable this feature, user has to enable `fast_lane` option on both the radios between which the fast lane must be created. To create a fast lane, use the following command:

```
uci set wireless.wifix.fast_lane='1'
```

Also, a user must set the `pref_uplink` option on only one radio that is preferred to be an uplink by using the following command:

```
uci set wireless.wifix.pref_uplink='1'
```

The radios having `fast_lane` option enabled will act as an enhanced independent repeater. In `iface_mngr_setup()`, it will be checked that in which radio `fast_lane` option has been enabled and accordingly set the operating mode (`iface_mngr_op_mode`) to 2. If the operating mode is 2, then as per the `pref_uplink` option script will classify the sub-interfaces (VAPs) into different groups. There will be two groups (`group0` and `group1`). If operating mode is 2 and `pref_uplink` is enabled, then AP VAP of that radio will be in `group0` and STA VAP of that radio will be in `group1`. Similarly if operating mode is 2 but `pref_uplink` option is not set, then AP VAP of the corresponding radio will be in `group1` and STA VAP will be in `group0`.

In this manner, the groups for fast lane are created at: `/var/run iface_mngr.conf`.

The configuration file is created in the following format:

```
mode=2 // 2 indicates fast lane mode
timeout=60// Timeout for the primary group STA to get connected to the
ROOT
group=1 ap_vap=ath1
group=0 sta_vap=ath11
group=0 ap_vap=ath2
group=1 sta_vap=ath21
```

Out of two groups, the group in which the STA VAP of the radio has the `pref_uplink` option set is considered as Primary group and the other group is considered as Secondary group.

- Initially, both the STA VAPs are scanning and the AP VAPs of both the groups are beacons.
- If a STA VAP becomes connected, then the application will check whether that STA VAP belongs to Primary group or Secondary group.
  - If STA VAP is from Primary group, bring up the AP VAPs that belong to Primary group if they are previously down and bring down all the STA and AP VAPs belonging to the Secondary group if they are previously up.
  - If STA VAP is from Secondary group, first, bring up the AP VAPs of Secondary group and then register a timeout to enable the STA that belongs to Primary group to scan and search for root AP before the timeout occurs. If STA from Primary group finds the ROOT AP and gets connected, then the preceding step is followed. Else, at the end of the timer, the application will bring down all the AP and STA VAPs that belongs to the Primary group.
- For DISCONNECT messages from `wpa_supplicant`, the application brings up only the STA VAP from other group:
  - When STA VAP from Primary group is connected, it signifies that all APs and STA VAPs from Secondary group are in down state. If the STA VAP from Primary group becomes

disconnected, then the application brings up the STA VAP from Secondary group. It is not necessary to bring up the AP VAP of Secondary radio because the AP VAPs on both the radios contain the same SSID.

#### Make command

```
make package/feeds/qca_10_4/qca-iface-mgr-10.4/{clean,compile} V=s
```

## 17.9 Configure WPS for range extenders using a single push button

The WPS hard push button control (PBC) is passed as a hotplug event to scripts/applications registered in /etc/hotplug.d/button/ with these environment variables set—'ACTION = pressed' and 'BUTTON = wps'.

It is mainly handled by the two scripts—50-wps and 52-wps-supplicant. They call the hostapd\_cli and wpa\_cli respectively to activate wps\_pbc on all APs and all STA (except proxy STAs for QWRAP) VAPs.

The other two scripts—wps-hostapd-update-uci and wps-suppliant-update-uci scripts—present in /lib/wifi/ handle the WPS events generated by hostapd and wpa\_supplicant respectively, and update the UCI setting, if required. For example, these scripts are employed when WPS handshake completes or when WPS handshake times out.

Another script, 54-wps-extender, is added to the /etc/hotplug.d/button/ to handle WPS enhancement for extenders. If the feature is disabled, the existing 50-wps and 52-wps-suppliant scripts pass the event to all existing VAPs and the newly added script is not executed. Else, if the feature is enabled, the existing 50-wps and 52-wps-suppliant scripts are skipped and the 54-wps-extender script is called.

This script passes the control to an application. The application parses the /etc/config/wireless file to find out VAPs to which the button push event needs to be passed. It then creates a list, in the ascending order of time, of WPS PBC activations and deactivations required, using the 'duration' and 'start\_time' arguments specified for those VAPs.

The application activates WPS PBC on VAPs, which have 'start\_time' as 0, and schedules a timer to wake up and process the next item of the list at the appropriate time. This is repeated for all the items in the list. WPS PBC activation and deactivation are performed using hostapd\_cli and wpa\_cli for AP VAPs and STA VAPs respectively. The WPS events from hostapd and wpa\_supplicant are handled by wps-hostapd-update-uci and wps-suppliant-update-uci scripts.

However, if this feature is enabled and the settings of the AP must be overwritten with those of the STA, the wps-suppliant-update-uci also updates the UCI settings of the AP VAPs. Additionally, the wps-suppliant-update-uci script calls 'wps\_config' command using hostapd\_cli to configure the new SSID and security settings on the AP VAPs.

The wps-suppliant-update-uci verifies the /etc/config/wireless file to identify all existing AP VAPs. This script also verifies the SSID suffixes from the /etc/config/wireless file and adds those suffixes to the SSID while configuring the AP VAPs.

Use this feature with ‘wps\_independent’ setting while running a single instance of hostapd. Otherwise, hostapd itself propagates the SSID and security credentials from one AP VAP to other AP VAPs, which conflicts with this implementation.

The application, which activates or deactivates WPS PBC on various VAPs, writes its PID to a file, which allows the wps-hostapd-update-uci and wps-supplicant-update-uci scripts to terminate it after a connection is successfully established.

**NOTE** Client fails to associate with AP when using WPS push button. WPS may not work when band steering is enabled. This is a known limitation, regardless of the use of primary and secondary ACLs. Primary and secondary ACL lists must be used in exclusive configuration for blacklisting and whitelisting. When band steering is enabled, primary ACL is used for blacklisting only.

## 17.10 Configure dynamic BSSID on repeater APs using wpa\_cli

This section describes how to set the dynamic BSSID on a repeater AP or range extender. Dynamic BSSID can be configured either using the wpa\_cli command or by editing the configuration file.

### 17.10.1 Set BSSID before a STA VAP connection

To set the BSSID using the wpa\_cli command before a STA VAP connection, enter the following command:

```
wpa_cli -p <path to ctrl sockets> bssid <network id> <BSSID>
```

For example, `wpa_cli -p /var/run/wpa_supplicant-ath01 bssid 0 00:03:7F:15:14:66`

### 17.10.2 Set BSSID in the configuration file

Alternatively, set the BSSID in the configuration file and reload the configuration file using the wpa\_cli command as follows:

1. Enter the following command to open and edit the file.

```
Vim /var/run/wpa_supplicant-ath01.conf
```

For example, in the following file, the `bssid=00:03:7F:15:14:66` line is the added entry to set BSSID dynamically.

```
network={  
    ssid="example"  
    proto=WPA RSN  
    key_mgmt=WPA-PSK WPA-EAP  
    pairwise=CCMP  
    group=CCMP  
    psk=06b4be19da289f475aa46a33cb793029d4ab3db7a23ee92382eb0106c72ac7bb  
    bssid=00:03:7F:15:14:66 }
```

2. After editing the file and saving it, enter the following command:

```
wpa_cli -p <path to ctrl sockets> reconfigure
```

For example, `wpa_cli -p /var/run/wpa_supplicant-ath01 reconfigure`

## 17.11 Automatic addition of Ethernet client to ProxySTA in QWRAP mode

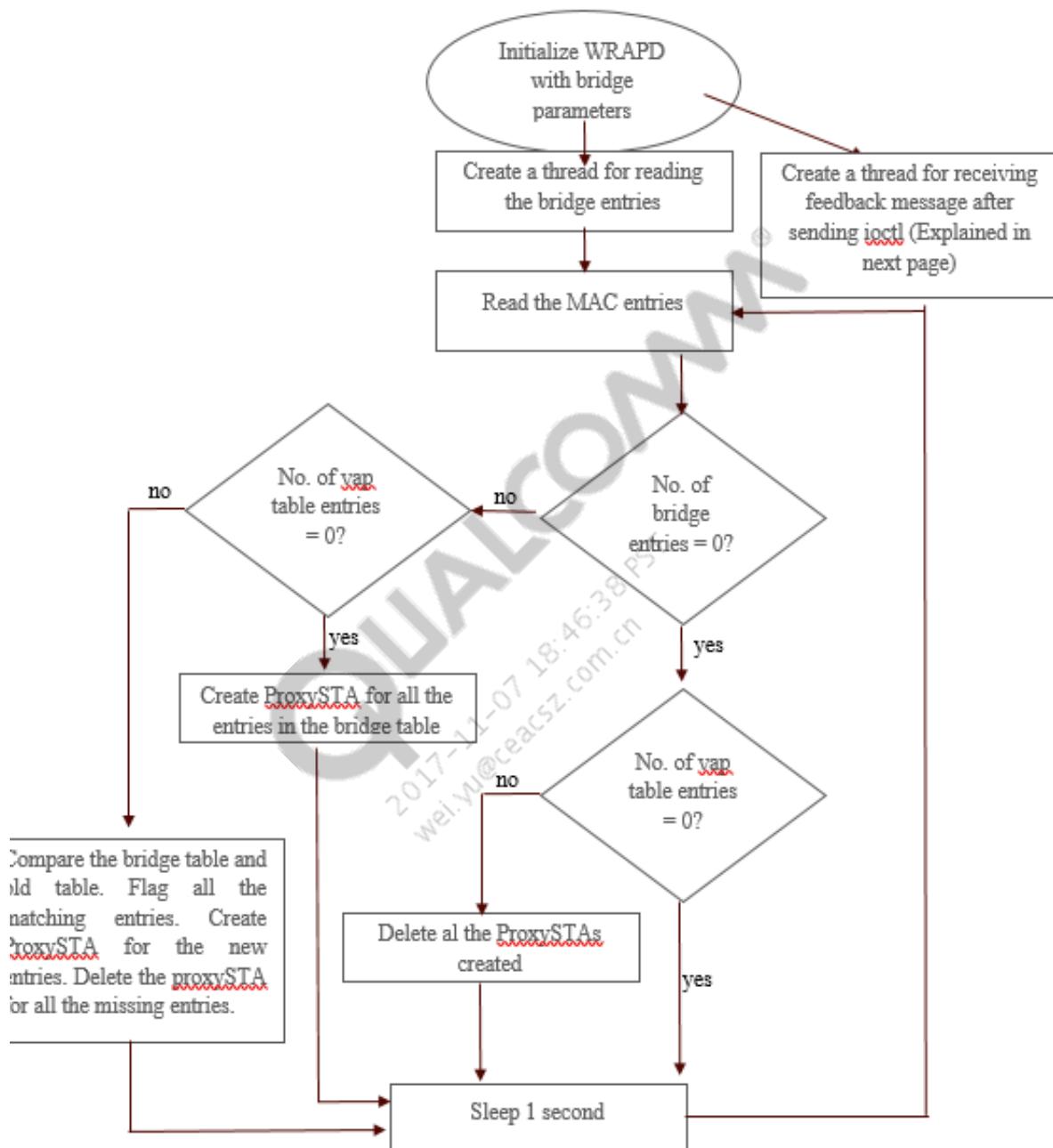
In QWRAP mode, the AP has to detect the clients behind it to create proxy stations. For a wireless station the AP detects them when they are associated with the AP. But for a wired station, the AP does not detect the wired clients as they are outside the WLAN driver. The wired stations are connected to the bridge which is outside the WLAN driver.

WRAPD application has the provision to add proxy stations manually. This requires human intervention each time a wired client is added. The AP needs to detect and create the proxy stations automatically without human intervention.

BRCTL is a software bridge that attaches the interfaces and forwards packets between them. The bridge intelligently learns the interface on which each mac address is connected with. It maintains a MAC address table and learning happens when frames are received. The bridge uses the MAC address table to forward the frames only to the appropriate interfaces. Whenever an entry is not present in the MAC address table, the frames are forwarded to all the interfaces (similar to broadcast).

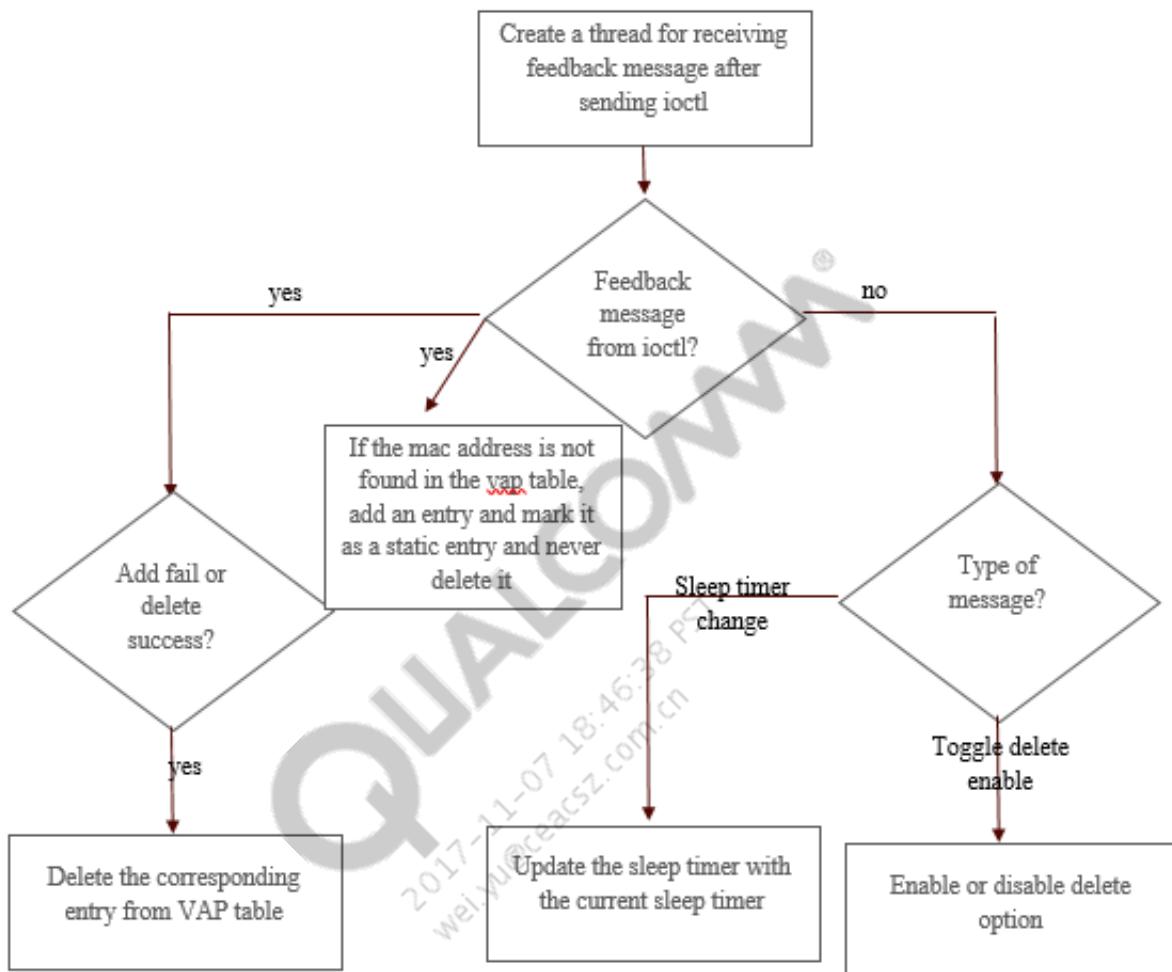
The bridge maintains a MAC table containing the MAC entries that can be reached from each interface. WRAPD app gets this list by reading the bridge entries in a fixed interval (ideally 1 second). The bridge entries are obtained from PROC entries or by passing an IOCTL (for old kernel). If any new entry is seen in the bridge MAC table, a proxySTA is created. If a proxySTA MAC address is not seen in the list, the proxySTA gets deleted (Deletion is optional and is configurable at runtime). Two threads are created for attaining the functionality. The main thread reads the bridge table, compares it with the existing Vap table and adds or deletes Vaps. The second thread is used for getting feedback message from the driver when vap creation is successful. This thread is also used to receive messages from another app for changing the parameters in runtime.

Flowchart — Main thread



Flowchart – Socket Feedback Thread

This thread receives the socket feedback message whenever there is an error in the VAP creation (ioctl error). The feedback message contains the mac address for which the ioctl was issued along with success or failure flag. This mac address is matched with the VAP table maintained and the entries for which the vap creation has failed are removed.



### 17.11.1 Configuration

The WRAPD app must be configured with bridge parameters for polling information on bridge MAC table. These parameters are specified when WRAPD app is started in Master mode.

- **-b**—Enter the bridge name (Eg. br-lan)
- **-i**—Enter the LAN interface name. Multiple interfaces can also be added by using this option multiple times (for example, eth1/eth0).
- **-t**—Specify the time in which the bridge mac table must be read. Default time is set as 1 second.
- **-I**—Specify the Wi-Fi interface on which the VAP has to be created (for example, wifi0).
- **-r**—Enable delete option

### 17.11.2 UCI configuration

To change the bridge name: (default is br-lan)

```
wireless.wifi0.qwrap_br_name=br0
```

To change the Ethernet interface name:( default is eth1)

```
wireless.wifi0.qwrap_eth_name=eth0
```

To change the wired station limit : (Default is 20)

```
wireless.wifi0.qwrap_sta_limit=10
```

To change the poll timer : (Default is 1 second)

```
wireless.wifi0.qwrap_poll_timer=5
```

To enable automatic addition feature: (it is disabled on default)

```
wireless.wifi0.qwrap_eth_sta_add_en=1
```

To enable automatic deletion feature: (it is disabled on default)

```
wireless.wifi0.qwrap_eth_sta_del_en=1
```

### 17.11.3 Limitations

- The ageing timer value should not be less than 30 seconds if the delete option is enabled. If the ageing timer value is set to be less than 30 seconds, repeated creation and deletion of vap will take place which will cause more cpu utilization.
- Wlanconfig should not be used to add or delete a proxySTA as wrappd doesn't come to know of it and thinks that the proxySTA is already added or deleted.
- Loops cannot be handled if delete option is enabled.
- The first frame from a wired client is lost, as proxySTA is not created at that point.

## 17.12 Enhanced Independent Repeater scan algorithm

- Enhanced Independent Repeater (EIR) performs scanning in background mode (that is, during scanning, radio switches between home channel and foreign channel).
- In the home channel, AP sends beacon and serves associated clients.
- In the foreign channel, AP stops sending the beacon and searches for root AP.
  - If foreign channel is an active channel, then AP sends a probe request to find the root AP.
  - If foreign channel is a passive channel, then AP listens for other AP beacons to find the root AP.
- Based on regulatory domain rules, channels are actively or passively scanned.

## Scan parameters

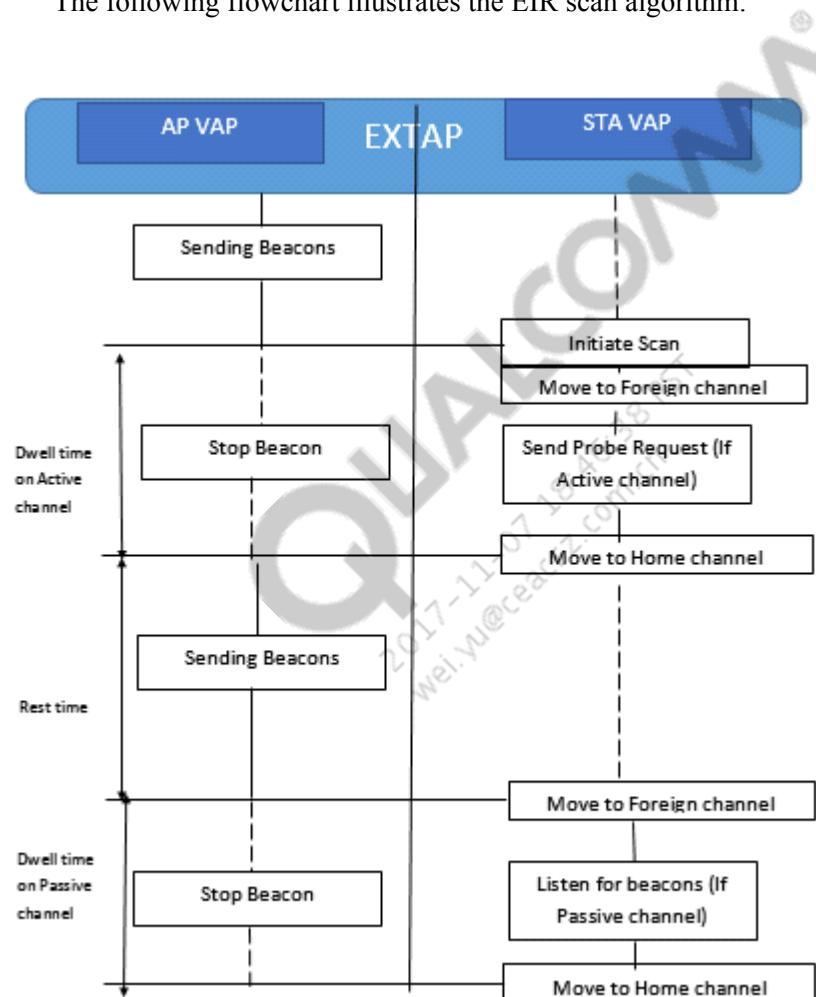
The rest time is for home channel and dwell time is for foreign channel.

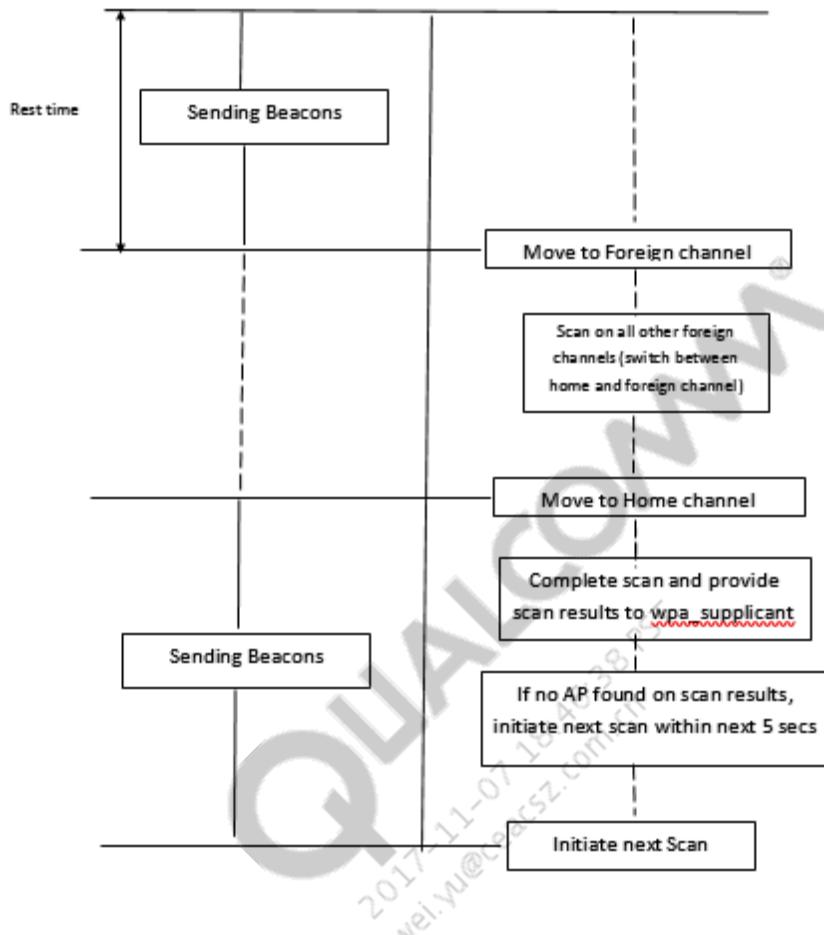
Rest time = 500 – 600 ms

dwell time on active channel = 150-200 ms

dwell time on passive channel = 130 -300 ms

The following flowchart illustrates the EIR scan algorithm:





## 17.13 Support for same SSID on root and repeater APs in star topology in QWRAP

The capability to support same SSID on a root AP and repeater APs is implemented. Repeaters can be connected to root AP in a cascading or star topology.

Before the implementation of this capability, when the same SSID was used on Root AP and repeaters, during Root AP down state, repeaters connected among one another (that is, STA VAP of range extender 1 (RE1) connects with AP VAP of RE2 and STA VAP of RE2 connects with AP VAP of RE1) and a connection in this fashion resulted in a packet loop. This problem is avoided with the support for same SSID on root and repeater APs in a star topology.

### 17.13.1 Design overview

Repeater includes a new IE (Extender IE) in beacon frame of all bands. Extender IE content is the same on beacons sent on all bands of a repeater. The Extender IE is of the following format:

IE number: vendor specific (221)

```

IE Length: 5
OUI: 8c-fd-f0
Vendor specific OUI type: 3
Extender info: 0x00 / 0xF0 / 0xFF

```

| Value of extender info | STA VAP connection status         | Root AP access           |
|------------------------|-----------------------------------|--------------------------|
| 0x00                   | No STA VAPs connected             | No Root AP access        |
| 0x0F                   | Reserved value                    |                          |
| 0XF0                   | At least one STA VAP is connected | No Root AP access        |
| 0xFF                   | At least one STA VAP is connected | Root AP access is there. |

The repeater includes all radio's AP VAPs MAC address in its association response frame. When this repeater (say RE1) connects to another repeater (RE2), RE1 receives this AP VAP's MAC address from association response and adds it into STA VAP connection list.

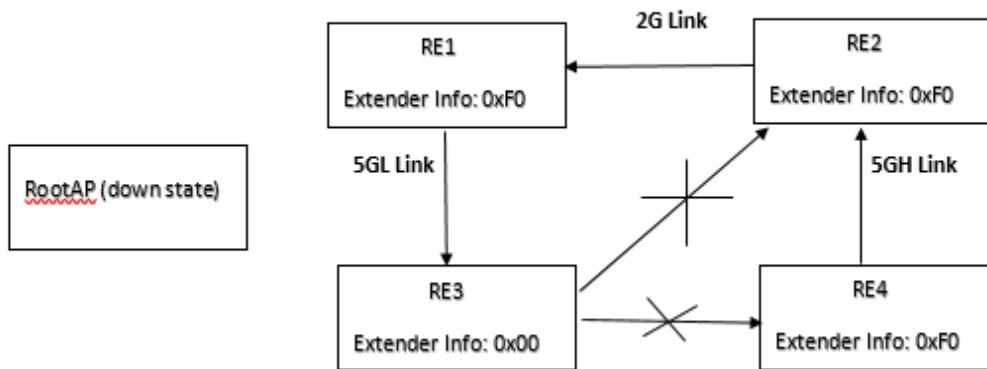
The repeater contains a global counter for the total number of repeater clients associated with all radios of RE.

### 17.13.2 Supported deployments

The following two configuration scenarios are supported with this design methodology.

#### Root AP in Down state

Consider the configuration scenario as illustrated in the following diagram.

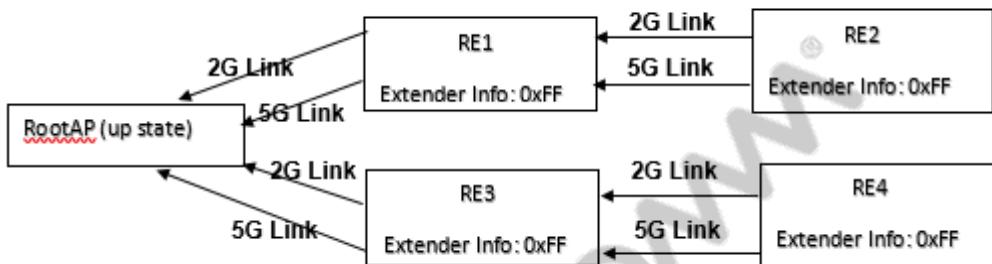


- When a root AP is in down state, all REs should have only one STA VAP connected. This is to make sure all REs are connected to each other and clients that are connected to repeaters can able to reach each other.
- Also, when one radio STA VAP is connected to another RE, other two radios STA VAP will scan for a root AP.

- If any repeater has one RE client connected on one of its radios, then that repeater cannot connect with any other repeater whose STA VAP is in connected state.

### Root AP in Up state

Consider the configuration scenario as illustrated in the following diagram.



- If any RE connects to a root AP or an RE that has root AP access, then all its connected repeater clients are disconnected. This behavior is necessary to detect new topology.
- If at least one STA VAP of RE1 connects with AP VAP of RE2, then make sure all RE1 STA VAPs are getting connected to RE2 AP VAPs. This can be accomplished by adding RE2 AP VAP MAC addresses in RE1 STA VAP connection list.
- RE1 receives RE2 AP VAP MAC address through association response frame.
- If any one STA VAP of RE connects to Root AP, then make sure other band STA VAPs of that RE also connects with the root AP.
- If any RE has scan entries for both root AP and another RE, then that RE must provide preference to the root AP during connection.
- If any RE has scan entries of more than one RE, then that RE must provide a preference to an RE whose RSSI value is high during connection.

### 17.13.3 Configuration

When any AP VAP and STA VAP are configured with same SSID in UCI configuration, this feature is enabled automatically. No manual configuration is needed to enable this feature.

### 17.13.4 Guidelines

- A maximum of up to three hops (Root AP->RE1->RE2->wireless client) are supported.
- Dynamically changing SSID during runtime is not supported.
- When all AP VAPs of root AP are in down state, all REs must have a maximum of one STA VAP connected.
- When all AP VAPs of root AP are in down state, RE1 STA VAP must not connect to RE2 AP VAP if RE2 STA VAP is already connected to RE1 AP VAP.

- When all AP VAPs of root AP are in down state, RE extender info should have value of 0xF0, if at least one RE STA VAP is connected.
- When all AP VAPs of root AP are in down state, RE extender information must have the value of 0x00, if no RE STA VAP is connected.
- When all AP VAPs of root AP are in down state, RE1 STA VAP must not connect to RE2 APVAP, if RE1 has any repeater clients associated and if RE2 has at least one STA VAP in connected state.
- When root AP is in up state, both bands of RE must connect to either the root AP or the same RE.
- When root AP is in up state, RE extender information must contain the value of 0xFF, if at least one RE STA VAP is connected.
- When root AP is in up state, RE extender info should have value of 0x00, if no RE STA VAP is connected.
- When only one AP VAP of root AP is in UP state and when only one STA VAP of RE1 associated with root AP, then the other STA VAP of RE1 must also scan only for root AP. The other STA VAP of RE1 must not associate with any other REs.
- When one STA VAP of RE1 associates with RE2, then other STA VAP of RE1 must also scan and associate to RE2 only.
- When root AP is in up state and present within the range, RE gives preference to root AP BSSID while connecting, instead of connecting to any other RE.

# 18 Voice and Multimedia Features

---

This chapter describes the capabilities supported for voice and multimedia traffic, such as Wi-Fi Multimedia (WMM) Flow Control and Buffer Management, video over wireless, voice enterprise, Smart Antenna, and Unified Smart Antenna API.

## 18.1 WMM Flow Control and Buffer Management

### 18.1.1 WMM Terminology

- AC (Access Category): A label for the common set of enhanced distributed channel access (EDCA) parameters that are used by a WMM STA to contend for the channel to transmit MSDUs with certain priorities. WMM defines 4 ACs.
- TID: Traffic Identifier; a 4-bit number that uniquely identifies a TSPEC.
- UP: User Priority; identical to the 3-bit priority subfield carried in the 802.1D Priority field.
- EOSP: End of Service Period is set by a WMM AP to 1 at the end of an unscheduled service Period (USP) and is set to 0 otherwise.

### 18.1.2 Theory of Operation

#### 18.1.2.1 WMM Procedure at an AP

An AP that supports WMM includes either a WMM Information Element or a WMM Parameter Element in every beacon. In response to a probe request, a WMM-enabled AP shall include a WMM Parameter Element in its probe response.

On receipt of an association request and subsequent transmission of a corresponding association response: the AP includes a WMM Parameter Element in the association response if the corresponding association request contained a WMM Information Element and treats the association as a WMM association. The same applies to a re-association request or re-association response.

If the destination address of a data frame to be transmitted on the wireless medium corresponds to a STA with a WMM association, the AP uses WMM QoS data subtype frame formats when transmitting the frame to it. If the destination address corresponds to a STA associated as a non-WMM STA, the AP does not use QoS subtype data frames.

|   |            |      |   |    |
|---|------------|------|---|----|
| 0 | Ack policy | EOSP | 0 | UP |
|---|------------|------|---|----|

**Figure 18-1 QoS Control Field**

The three-bit UP field carries the priority bits of the 802.1D Priority and is used to signal the priority for this frame. It also implies the sending AC according to the mappings in 802.1D Priority to AC Mappings. The UP for each MPDU of a MSDU is the same value.

**Table 18-1 802.1D Priority to AC Mappings**

| Priority | 802/1D Priority | 802                | Access Category | WMM Designation |
|----------|-----------------|--------------------|-----------------|-----------------|
| Priority | 802.1D Priority | 802.1D Designation | Access Category | WMM Designation |
| Lowest   | 1               | BK                 | AC_BK           | Background      |
|          | 2               | -                  |                 |                 |
|          | 0               | BE                 | AC_BE           | Best Effort     |
|          | 3               | EE                 |                 |                 |
|          | 4               | CL                 | AC_VI           | Video           |
|          | 5               | VI                 |                 |                 |
|          | 6               | VO                 | AC_VO           | Voice           |
| Highest  | 7               | NC                 |                 |                 |

### 18.1.2.2 WMM TID Classification

WMM maps the TOS (type of service) field present in the 802.3 packet from the IP stack to the WLAN access categories as mentioned in [Table 18-1](#) and [Figure 18-2](#).

The macro TID\_TO\_WME\_AC converts the given TID to the respective access category.

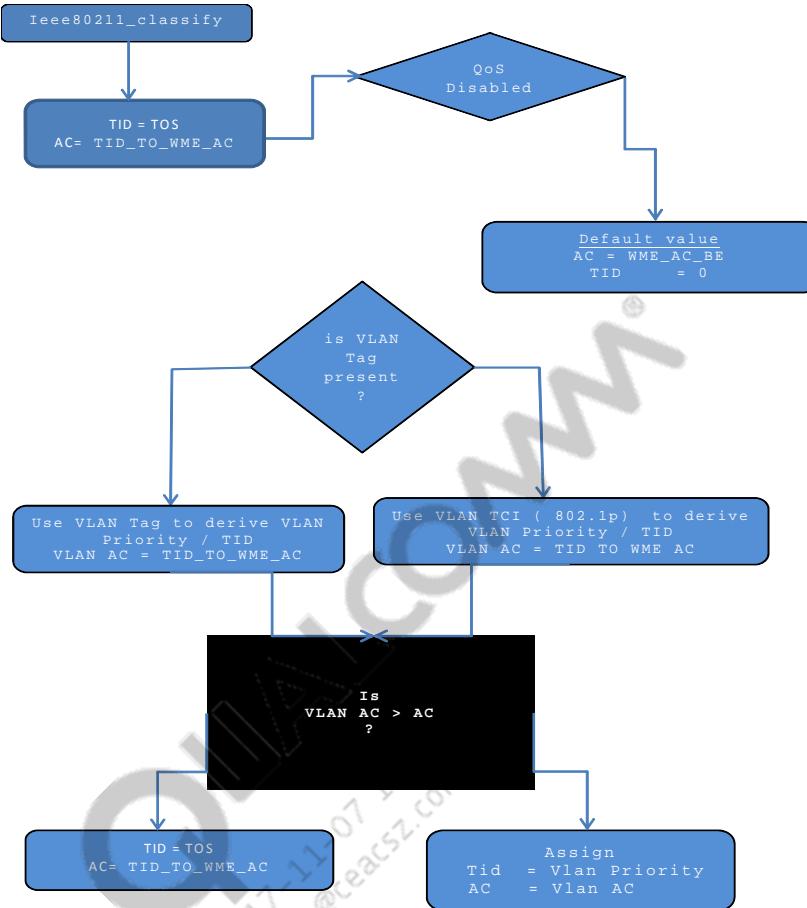
**Table 18-2 TID to AC Conversion**

| TID  | Access Category       |
|------|-----------------------|
| 0, 3 | WME_AC_BE             |
| 1, 2 | WME_AC_BK             |
| 4, 5 | WME_AC_VI / WME_AC_VO |

When QoS is enabled, the default priority is set to WME\_AC\_BE.

When a VLAN is present, a VLAN tag associated with the packet can be used to derive VLAN priority. This VLAN priority is directly mapped to the TID.

If a VLAN tag is not present and the packet is operating under IEEE 802.1p, then the VLAN priority is derived from the VLAN TCI field.

**Figure 18-2 Packet Classification**

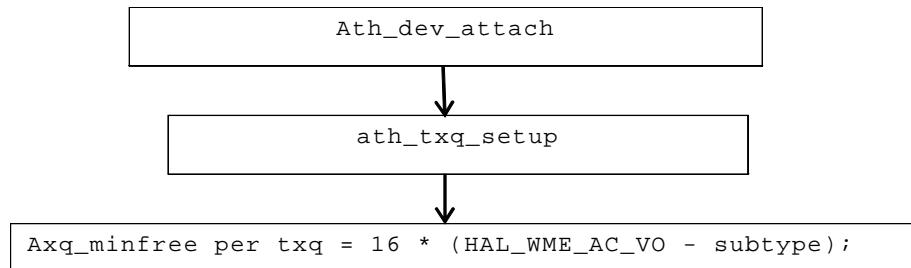
Based on the Access Category derived using above flow chart, frames are queued to corresponding hardware queues.

### 18.1.3 Implementation

#### 18.1.3.1 Queue Management for Transmission

The transmit queues for the QoS data with a different Access Category (AC) and for the special type of data such as CAB, XR or UAPSD are allocated and set up during `ath_dev_attach`, by claiming the available queue within `sc_txq[]` whose `tqi_type` equals `HAL_TX_QUEUE_INACTIVE` (indicating that this queue is currently unused), or simply by claiming the reserved queue for beacon, CAB, or PSPOLL data. Once allocated, the `tqi_type` of the claimed queue and the corresponding bit of the bitmap `sc_txqsetup` are masked. Note that there is no buffer or descriptors being linked to the claimed queue during this phase.

`axq_minfree` of the hardware transmit queue is initialized in the following way:

**Figure 18-3 axq\_minfree Initialization**

### 18.1.3.2 WMM Queue Flow Control

One hardware transmit queue is maintained per Access Category. Packets sent from above layers are assigned to queues based on their priority as explained in [WMM TID Classification](#).

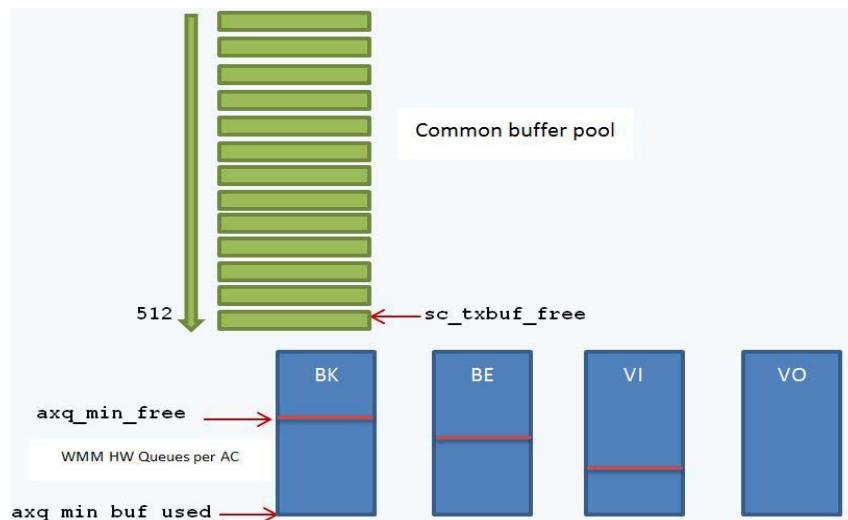
WMM Flow control is achieved with the presence of two members, *axq\_minfree* and *axq\_num\_buf\_used*. The former represents the number of free tx\_bufs required in the common buffer pool, and the latter represents the number of used tx\_buf for that queue.

To avoid low priority traffic from hogging when high priority traffic is flooding, MIN\_BUF\_RESV macro defined as 16, which represents the number of buffers reserved per AC used in the below mentioned method.

*sc\_txbuf\_free* holds the number of free tx buffers for a particular device (radio).

If *axq\_num\_buf\_used* is greater than MIN\_BUF\_RESV and *sc\_txbuf\_free* is less than *axq\_minfree*, then update the stats as “txbuffer not available” in *sc\_stats.ast\_txq\_nobuff[txctl->qnum]* and return without transmission of that frame. The caller is responsible for dropping that frame.

#### Buffer management with WMM Flow control

**Figure 18-4 WMM Flow Control Queue Initialization**

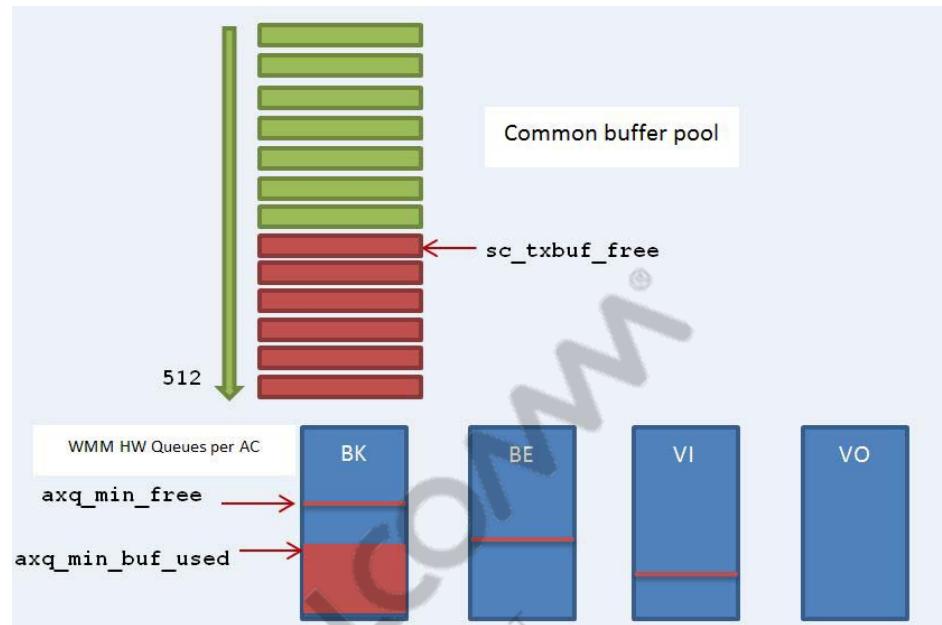


Figure 18-5 WMM Flow Control During Transmission

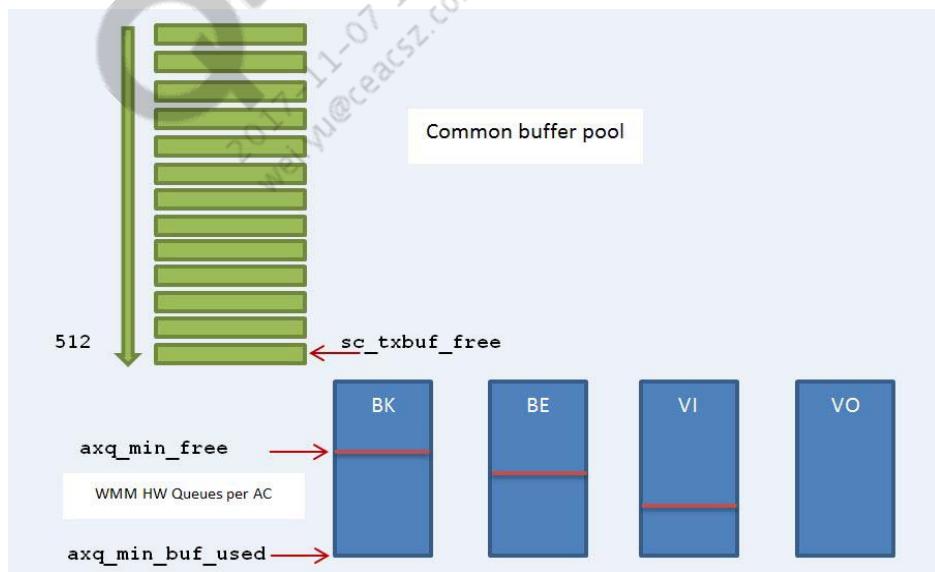
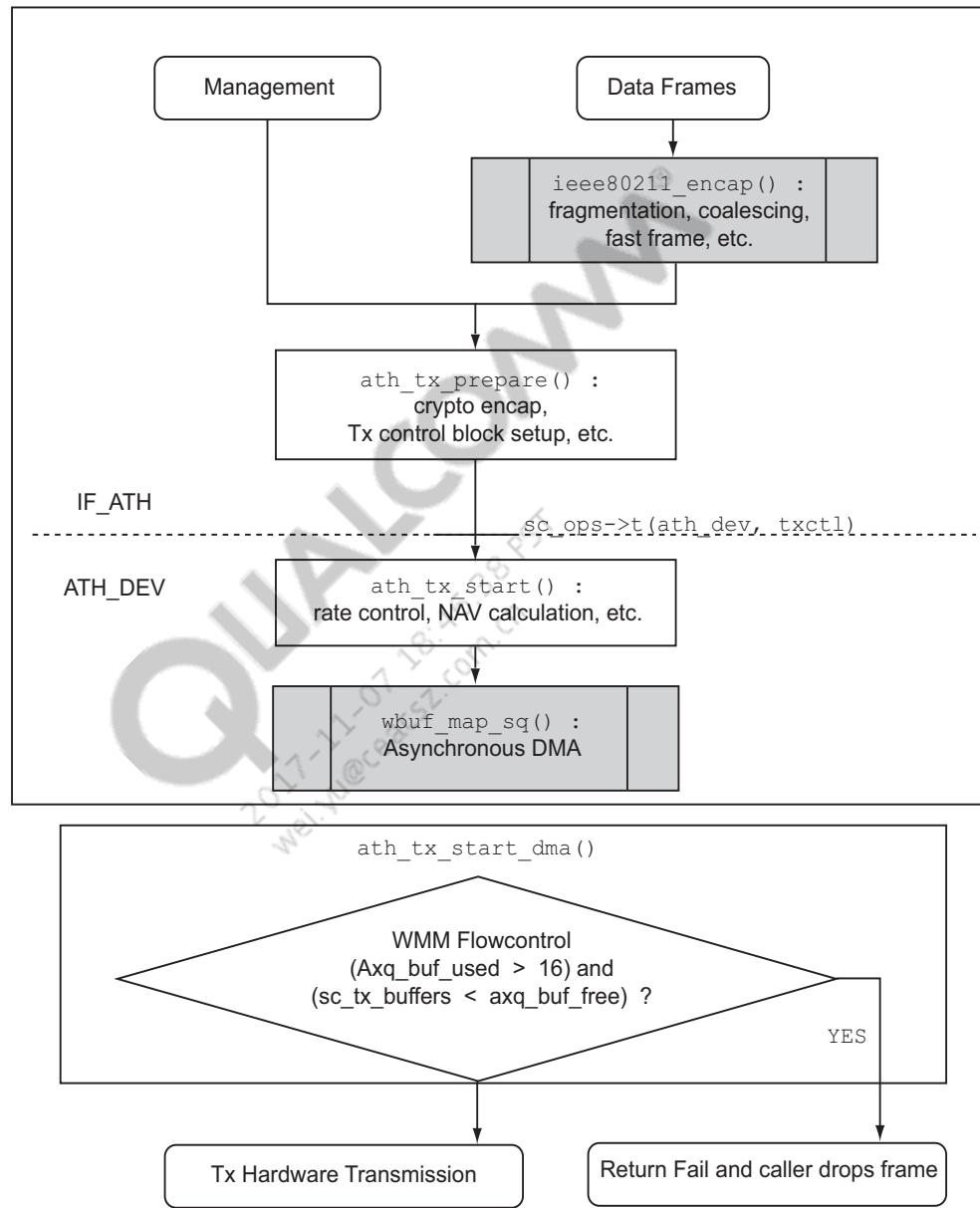


Figure 18-6 WMM Flow Control After Transmission Completion

### 18.1.3.3 Tx Flow

The basic transmit flow when WMM is involved is illustrated by [Figure 18-7](#).

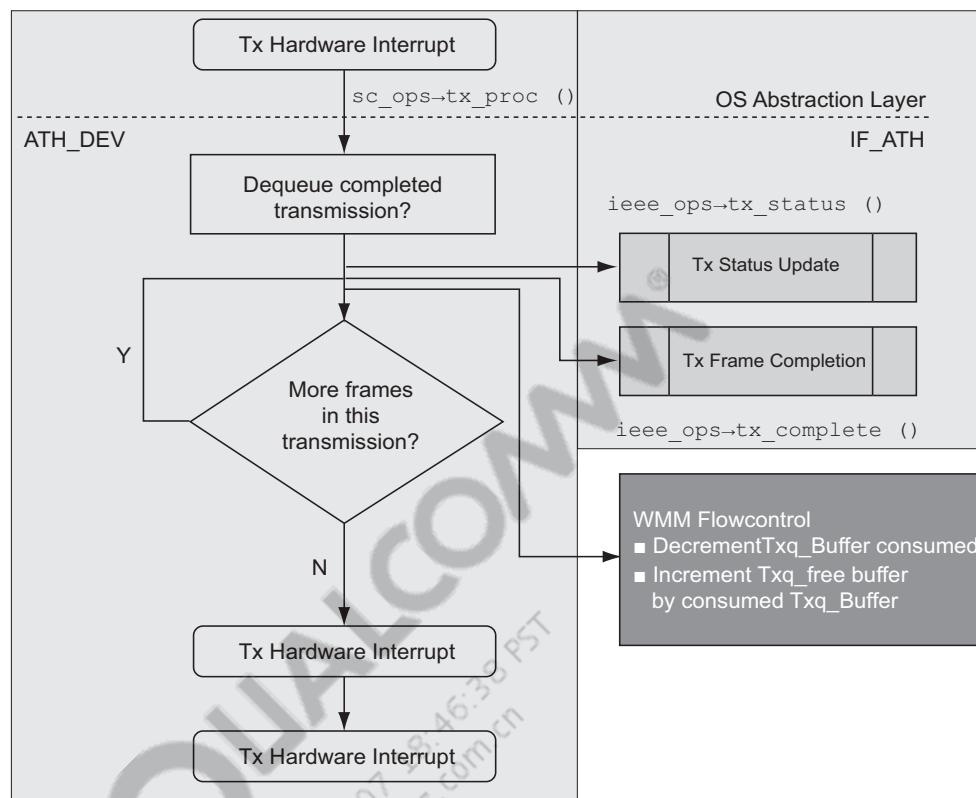


**Figure 18-7 Tx flow with WMM Flow Control**

### 18.1.3.4 WMM & Tx Completion Flow

The basic transmit completion flow is illustrated by [Figure 18-8](#).

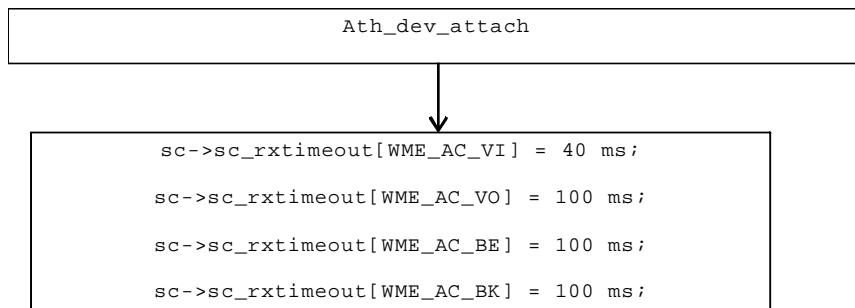
Tx completion, along with other stats, also updates the WMM parameters `axq_num_buf_used` by decrementing and `sc_txbuf_free` by incrementing.



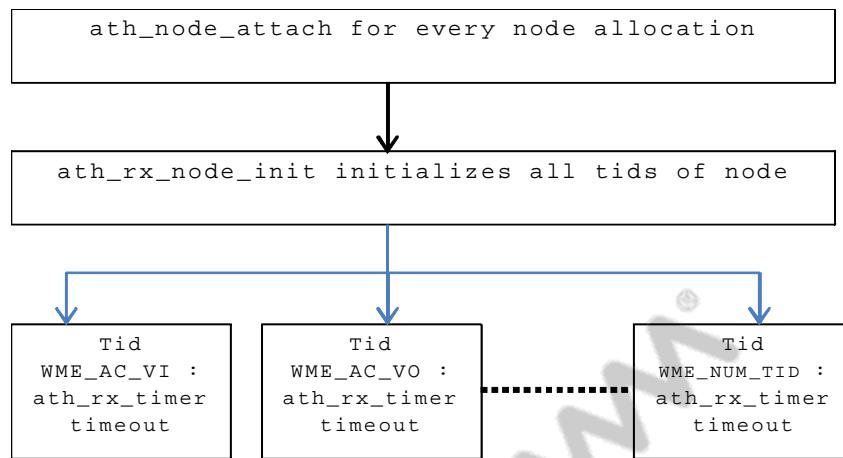
**Figure 18-8 Tx Completion with WMM Flow Control Buffers Update**

#### 18.1.4 WMM & Receive Path

Although the Rx path does not involve any major WMM component, ath\_rx\_timer is noted here. ath\_rx\_timer is responsible for handling received sub-frames, and it is initialized for every TID and for every Access Category. WMM Access Category timers are initialized as described in [Figure 18-9](#) and [Figure 18-10](#).



**Figure 18-9 ath\_rx\_timer Timeout Initialization**

**Figure 18-10 Node Rx Initialization for WMM**

The WMM QoS unicast data frame needs special aggregation processing: add the received buffer to the receive BAW, and add to the BAW buffer rxtid>rdbuf. Indicate all in-order received frames to upper protocol layer, and schedule an Rx timer to flush the remaining pending received buffers.

## 18.1.5 Configuration

### 18.1.5.1 User configuration

```

iwpriv athX setwmmparams CWMIN CWMAX AIFS TXOP ACM NOACKPOLICY
iwpriv athX getwmmparams CWMIN CWMAX AIFS TXOP ACM NOACKPOLICY
iwpriv athX wmm 1 (enables WMM)
iwpriv athX wmm 0 (disables WMM)
iwpriv athX get_wmm (displays current status of WMM)
    
```

Enable TOS for the packets coming out of Network Protocol Stack. For Windows OS, add the following entry to the Windows registry:

```

HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Add REG_DWORD name "DisableUserTOSSetting" as 0
    
```

## 18.2 Voice Enterprise

Voice Enterprise is a WiFi initiative to support voice applications and voice-enabled devices in enterprise WiFi networks. Typical enterprise networks consist of RADIUS infrastructure for security with multiple APs and STAs that carry a wide range of traffic including data, audio and video. Accordingly, multiple simultaneous voice calls must co-exist with several types of traffic and yet ensure adequate voice call quality in terms of latency, jitter, and packet loss performance. Medium access is prioritized in compliance with WMM and voice traffic is sent using AC\_VO priority. WMM also provides for effective power-save procedures to prolong battery life of mobile devices. Furthermore, voice-enabled mobile devices that roam between APs that are themselves handling other voice traffic must do so in a seamless fashion without any degradation.

Relevant aspects of IEEE 802.11k, 802.11r and 802.11v are implemented to meet compliance with WiFi Voice Enterprise requirements. These include:

- Radio Resource Measurement and Reporting (IEEE 802.11k)
  - Beacons and probe responses contain a radio measurement bit in the Capabilities IE to indicate feature support, RRM Enabled Capabilities IE to indicate support for different types of measurements, Country IE, Power Constraint IE, WFA Vendor Specific TPC Report IE.
  - Beacons and probe responses contain BSS Load IE and BSS Available Admission Control IE to indicate current BSS load in terms of channel utilization and admission capacity.
  - Beacons contain Quiet IE specifying a quiet period and a quiet duration; this is used by an AP to make all STAs in the BSS silent to let the AP make radio resource measurements of the RF conditions.
  - Beacon reports for AP infrastructure to monitor WLAN coverage; an AP sends a beacon report request to STA(s) to learn about the list of APs the STA(s) can hear on one or more channels and report back with a beacon report response.
  - Neighbor reports to provide hints to STA(s) about potential roaming candidates for BSS transition. STA(s) send a neighbor report request to the associated AP which in turn respond with a neighbor report containing a list of APs.
  - Traffic Stream/Category Measurement (TSM) to determine the quality of an ongoing traffic stream between the AP and a STA and the report provides the transmit-side performance metrics for the measure traffic stream based on trigger conditions.
  - Power save procedures for Radio Measurement Action frames so that these frames are buffered when a STA is in power save mode.
- Fast BSS Transition (IEEE 802.11r)
  - Beacons and probe responses contain Mobility Domain IE to indicate support
  - Fast BSS Transition (FT) procedures include initial admittance into a FT mobility domain and subsequent over-the-air or over-the-DS re-association to target AP using FT protocol authentication.
- Wireless Network Management (IEEE 802.11.v)
  - BSS transition management enables an AP to request STA(s) to transition to a specific AP or to recommend STA(s) to transition to a preferred list of APs either for better load balancing or for current BSS termination.

### 18.2.1 Implementation

Three new modules, *rrm*, *wnm*, *admctl*, have been added to support the Voice Enterprise functionality in addition to changes in hostap, UMAC, and LMAC.

The functionality in *rrm* includes generation of beacon measurement requests; processing of received neighbor report requests and respond with neighbor reports, generate TSM requests and link measurement requests. Similarly, *wnm* provides support for sending BSS Transition management request and responding to a BSS Transition management query with a neighbor report. The functionality in *admctl* includes processing of TSPEC requests from STA(s) on a per-

AC basis and managing power-save behavior of STA(s) if the TSPECs supersede default power-save behavior for the AC. Additionally, *admctl* keeps tab of the medium time occupied by a STA and makes sure that the STA does not violate the TSPEC compact.

The functions *ieee80211\_add\_rrm\_cap\_ie*, and *ieee80211\_bssload\_beacon\_setup* add the RRM Enabled Capabilities IE and BSS Load IE to beacons and probe responses. Currently, the admission capacity in the BSS Load IE is not being computed. Similarly, the function *ieee80211\_quiet\_beacon\_setup* adds the Quiet IE to beacons. The Quiet period and quiet duration are set statically at compile time as defined by *WLAN QUIET\_PERIOD\_DEF* and *WLAN QUIET\_DURATION\_DEF*.

The functions *wlan\_send\_beacon\_measreq*, *wlan\_send\_tsm\_measreq*, *wlan\_send\_link\_measreq* send measurement requests using action frames to associated STA(s).

On receiving a neighbor request action frame from a STA, the AP will call *ieee80211\_fill\_nrinfo* and *ieee80211\_add\_nr\_ie* to fill the neighbor report based on the AP scan entries with the requested SSID and send back a neighbor report as a response.

In response to a BSS Transition Management query request action frame from a STA, the AP will call *ieee80211\_send\_bstm\_req* which will call *ieee80211\_fill\_nrinfo* and *ieee80211\_add\_nr\_ie* to fill the neighbor report based on the AP scan entries with the same SSID.

Fast BSS Transition is implemented in hostap with minimal support in the driver which includes allowing IEs from hostapd to be included in beacons and probe responses, forward authentication/association request frames to hostapd, receive authentication/association responses along with optional IEs from hostapd and send them over the wireless medium. Furthermore, hostapd is allowed to create a node for a STA to support Fast BSS Transition over-the-DS.

The functionality in *admctl* includes addition/deletion of TSPECs, calculation of medium time, and policing the admitted traffic streams for admitted medium time. Policing of STAs for admitted time is currently not yet supported.

## 18.2.2 802.11k

Currently, the driver software contains a partial implementation of the 802.11k specification. In this implementation the access point attempts to gain information of the environment from the connected client by sending various messages to it and receiving responses. This feature is required for effective handling of BSS by the AP. The access point may discover the load on a particular channel and neighbor access points present or operating on other channels. In this implementation supporting channel load, various statistics, neighbor report, are currently supported.

### 18.2.2.1 Operation

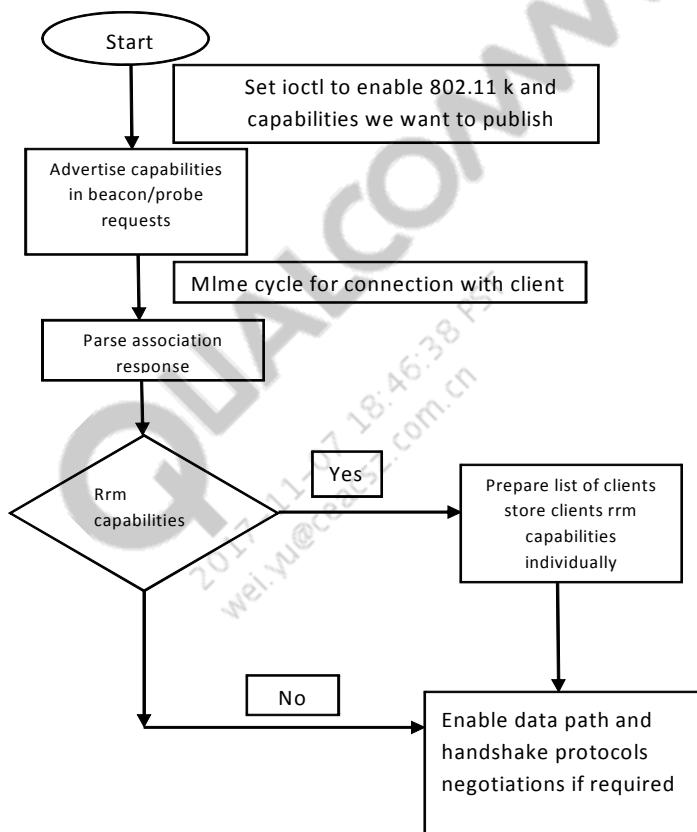
This module is embedded into the UMAC module and performs a BSS management related function by sending different commands. Various requests are sent to various connected clients. No specific hardware is required to implement this feature on the AP; the feature can work on any platform on which a VAP can be created.

Various requests supported by this 802.11k implementation are:

- Channel load
- 802.11k statistics-related request
- Neighbor request
- Noise histogram

In the current implementation there is a dependency on wifirm and the athadhoc user space application to send and receive commands respectively.

[Figure 18-11](#) provides a high level view of the 802.11k implementation.



**Figure 18-11 802.11k High Level Diagram**

### 18.2.2.2 Implementation

- No Software APIs implemented
- No hardware blocks directly accessed by software
- User CLI command APIs for configuration and control
- CLI commands
 

```
iwpriv interface ath0 rrm 1 (starts module)
      iwpriv interface get_rrm (gets rrm status)
```

**NOTE** Extensive use of this feature may result in a decreased station and AP throughput.

### 18.2.2.3 Configuration

To start this module, execute the commands described in previous section. Various measurement related requests can be shown on a sniffer, and usage of these requests can be explained.

#### Channel measurement request

The following requests assume 11k is enabled by the CLI commands previously mentioned.

- **Wifirrm interface name:** sendbcnrpt dest mac bssid chan num reg class
  - **dest mac address:** MAC address of associated station to which to send beacon request.
  - **Bssid:** BSSID of desired AP (RSSI to be determined)
  - **Chan num:** chan number for which statistics are to be determined
  - **Reg class:** regulatory class of the operating channel.

Upon reception of a beacon request, the station scans on the channel specified in the request and responds with the RSSI of the matching BSSID and sends the response.

**NOTE** If BSSID is set to broadcast (ff:ff:ff:ff:ff:ff) BSSID, then the station reports the best AP (in terms of RSSI) on that channel to the AP.

#### Channel Load

- **Wifirrm interface name:** sendchload cmd reg class destmac channel
  - **cmd:** reserved for future; on current implementation it should be passed as any positive value other than zero.
  - **reg class:** regulatory class of operating channel
  - **destmac:** MAC address of associated station
  - **channel:** channel on which station will calculate channel load

#### 802.11kStatistics\_NbStations

- **Wifirrm interface name:** sendstastats dst mac duration gid
  - **dst mac:** MAC address of associated client
  - **duration:** length of time to receive statistics. Value is in milliseconds
  - **gid:** group ID; this value is taken from the 802.11k specification, for these two requirements the value should be 10.

#### 802.11kStatistics\_TxFrameCount

- **Wifirrm interface name:** sendstastats dst mac duration gid
  - **dst mac:** MAC address of associated client
  - **duration:** desired interval for statistics. Value is in milliseconds
  - **gid:** group id; this value is taken from 802.11 k specification, for these two requirements the value should be 0.

### **802.11kStatistics\_AckFailure**

- **Wifirm interface name:** sendstastats dst mac duration gid
  - **dst mac:** MAC address of associated client
  - **duration:** length of time to receive statistics. Value is in milliseconds.
  - **gid:** group id; this value is taken from 802.11k specification, for these two requirements the value should be 1.

### **802.11kStatistics\_AccessDelayVoice**

- **Wifirm interface name:** sendstastats dst mac duration gid
  - **dst mac:** MAC address of associated client
  - **duration:** length of time to receive statistics. Value is in milliseconds
  - **gid:** group ID; this value is taken from 802.11k specification, for these two requirements the value should be 10.

### **802.11kReport\_JammingLevel**

- **Wifirm interface name:** sendnhist dstmac duration reg class channel
  - **dst mac:** MAC address of associated client
  - **duration:** length of time to receive statistics. Value is in milliseconds
  - **reg class:** regulatory class of operating channel
  - **channel:** channel on which station will calculate channel load

## **18.2.3 Source Code**

### **WNM**

```
umac/wnm/ieee80211_wnm.c
umac/wnm/ieee80211_wnm_ap.c
umac/wnm/ ieee80211_wnm_priv.h
```

### **RRM**

```
umac/rrm/ieee80211_rrm_ap.c
umac/rrm/ieee80211_rrm.c
umac/rrm/ieee80211_rrm_ie.c
umac/rrm/ieee80211_rrm_priv.h
```

### **ADMCTL**

```
umac/admctl/ieee80211_admctl_ap.c
umac/admctl/ieee80211_admctl.c
umac/admctl/ieee80211_admctl_priv.h
```

### **MLME**

```
umac/mlme/ieee80211_bssload.c
umac/mlme/ieee80211_bssload.h
umac/mlme/ieee80211_quiet.c
```

```
umac/mlme/ieee80211_quiet.h  
umac/mlme/ieee80211_mlme_app_ie.c  
umac/mlme/ieee80211_mlme_btamp.c
```

## 18.3 Video over Wireless

The Video over Wireless implementation is intended to provide features and fixes to the existing wireless driver. Most of these fixes and features are in transmit path of the wireless driver. These address mostly rate control but not limited to it. The intent is to reduce the packet loss at wireless receiver. These fixes are either WAR for current hardware issues or fixes in software.

The following Video over Wireless fixes/features are implemented in the current release.

- Transmit buffer enhancements
- Retry reordering improvements
- Retry time reduction – aggregate size control
- Flow control
- Video stream protection
- Rate control improvements (RCA)

### 18.3.1 Acronyms

- PER – Packet Error Rate
- PLR – Packet Loss Rate
- VSP – Video Stream Protection
- Traffic Class – Priority mapping derived from DSCP value of IP datagram or VLAN priority of the frame. Usually this is one of the following:
  - BE – Best Effort priority traffic class
  - BK – Back Ground priority traffic class
  - VI – Video priority traffic class
  - VO – Voice priority traffic class
- wifiX – Either wifi0 or wifi1
- CBR – Constant Bit Rate
- UBR – Unspecified Bit Rate
- HTB – Hierarchical Token Bucket. Linux traffic shaping queue discipline, which can guarantee the CBR for one kind of traffic class, while maintaining the UBR for other traffic classes. Note that CBR and UBR are more Layer 2 throughput measures.

## 18.3.2 VoW Features

### 18.3.2.1 Transmit Buffer enhancements

In most cases the packet loss is caused at the ingress of the wireless access point. The driver transmit path requires a descriptor to queue the buffer to the hardware. In the current implementation the number of transmit descriptors varies from board to board based on the memory available. In general most boards have 512 transmit descriptors. These buffers would be sufficient for best effort traffic, where the driver can drop a buffer in presence of a ‘no-descriptor’ case. But for the video case, the driver should make sure that it does not drop the video packet. To achieve this there are two enhancements made:

- Increase the number of descriptors from 512 to 1536

Though this cannot guarantee the buffer for every frame, in the presence of a burst of traffic, the driver can make sure that it has enough buffers to hold the video traffic. This would be helpful in multiple client scenarios.

This increased number of buffers is configured only when ATH\_SUPPORT\_VOWEXT is defined through build system. ATH\_TXBUF configuration variable is defined in *build/scripts/board\_type/config.board\_type* file. For example, for *ap113*, modify *build/scripts/ap113/config.ap113*.

- In the presence of mixed traffic, reallocate the lower priority packet buffer to a high priority buffer

In the presence of mixed traffic, where BE/BK traffic and VI traffic are present, there is a chance that a station receiving BE/BK traffic could be at a lower RSSI than the station at receiving VI traffic. This can cause BE/BK traffic to consume more number of descriptors than the VI traffic. In such a scenario the driver might see a ‘no-buffer’ condition to send a VI frame. If this feature is enabled, it would de-queue one single buffer from the BE/BK traffic and assigns to VI traffic, so that priority VI traffic continue to get buffers.

This algorithm works only when ATH\_CAP\_VOWEXT\_FAIR\_QUEUING is enabled.

### Implementation

This is implemented through the function *ath\_deq\_single\_buf()* in *ath\_xmit.c*. Refer to function *ath\_tx\_start\_dma()*. The above function walks through the BE/BK transmit queues, and frees the frame from tail, only if the frame is not already tried once.

### Configuration

- **Compile time**

This requires the ATH\_SUPPORT\_VOWEXT configuration macro defined in build system. Please see above file for build configuration file.

- **Run time configuration**

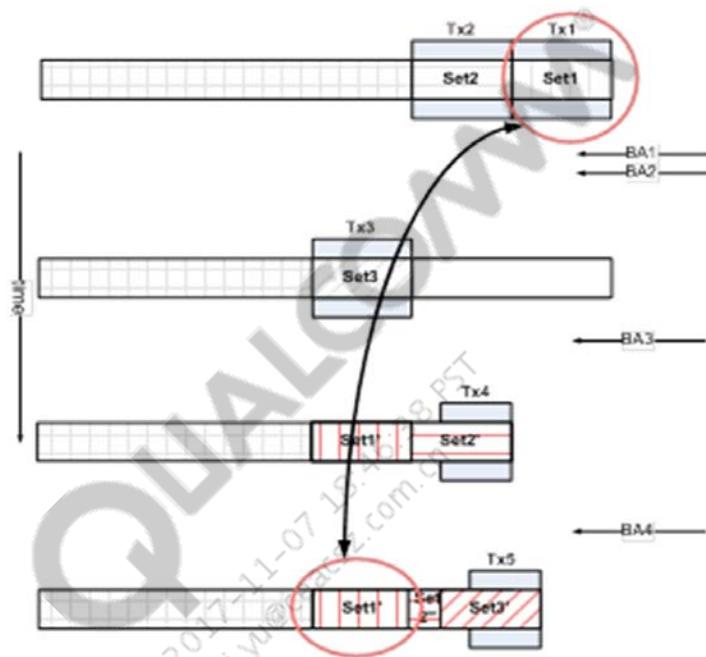
If VoW is compiled, it would be enabled by default. To further enable or disable, clear the ‘0’ bit in the VowExt configuration.

```
iwpriv wifiX setVowExt 28
```

Refer to the VSP section for de-queueing VI frames in the presence of video congestion.

### 18.3.2.2 Retry reordering improvements

For better performance, drivers can queue up to two aggregates to the hardware. The aggregate can either be completely successful, or there could be a few frame failures. When frames fail, they will be added to the head of the pending frames queue. See [Figure 18-12](#).



**Figure 18-12 Retry Reordering**

In [Figure 18-12](#), if there are two aggregate sets are queued to hardware, the failed frames from TX1 would be processed by software while hardware is busy sending TX2. The software also tries to queue TX3 before processing the failed frames of TX1. The failed frames from TX1 would go just before TX4. This might cause receiver reorder buffer to time out and drop the retried frames or new sequences would be waiting for old sequences, adding more delay and jitter.

There are two different solutions. The first is to limit the hardware queue depth to one always; the second is to send the frames always in correct order. While the first one is addressed in a different algorithm (retry delay reduction), retry reordering addresses the second case. The algorithm is to sort the frames in ascending order of sequence numbers. If VoW is enabled, this is done for all traffic classes.

### Implementation

The implementation of this feature is in function `ath_tx_complete_aggr_rifs()` in `ath_xmit_ht.c`. Algorithm is implemented in function `ath_insertq_inorder()`.

## Configuration

- Compile-time configuration

Enable ATH\_SUPPORT\_VOWEXT in build/scripts/<board>/config.<board>

- Run-time configuration

Check the current configuration of the VoW. If bit 8 is ‘1’ it is enabled already. To disable, place ‘0’ in bit 8; to enable, place ‘1’ in bit 8. For example, if the current configuration returned is 1, using below command would enable this algorithm.

```
iwpriv wifiX setVowExt 0x9
```

### 18.3.2.3 Retry time reduction and aggregate sizing

Over many experiments it has been observed that the frames at the tail of the aggregate seems to be lost more often than the frames at head. So the frames at end of the aggregate would experience more retries. In specific implementations, the frames that experience a delay of more than 40 milliseconds would be discarded at the receiver. The number of failures at tail of the aggregate would contribute to more PER and rate drop at the transmitter. Each frame could be retried either by hardware or software. When the frame loss is huge, the hardware retries further increases the delay and causes more loss. Also the way the RTS and retries occur is chosen by the rate adaptation mechanism.

The purpose of this methodology is to use the information in aggregate subframes to choose the best rate, best retry series and aggregate sizes, so that the frame loss can be minimized.

## Theory of operation

As mentioned above, the receiver drops the packets that arrive with a delay of more than 40 ms. Contribution to the huge delay could be due to two reasons

1. The medium is very busy or noisy and there are too many aggregate failures, resulting in too many retried frames.
2. Internal queuing delays due to software queuing policies.

This VoW feature tries to address the first case of the problem; that is, try to reduce the time taken by a frame for transmission by controlling the aggregate size and hardware retries. This tries to minimize the peak retry delay at the transmitter hardware. This addresses the issue in two parts

1. Limit the number of hardware retries & stop the double buffering of packets at hardware transmit queue
2. Reduce the aggregate size
  - Limit the hardware retries and double buffering

All Qualcomm Technologies chips can have multiple transmit units queued at any time. The transmit unit is defined to be one transmittable unit on air. In 802.11a/b/g mode the transmit unit is one single frame and in 802.11na/ng mode it is one aggregate. The current hardware queuing depth is two such units.

Double buffering occurs in hardware if there is more than one transmittable unit queued to hardware transmit queue at any given time.

- Limit the hardware retries and stop double buffering if
  - TailErrors >= MAX(2 \* head errors, 4) and aggregate-size > 8
 OR
  - Instantaneous error rate is more than 50% and aggregate size is not less than 8.
 OR
  - If the delta between the current transmit duration timestamp and the first tried transmit timestamp is more than 5 ms.
 OR
  - (d) Hardware retry/double buffering has been limited in any of the last 10 aggregates.
- Reducing the aggregate size
 

Transmit aggregate sizing will be reduced as follows

$$\text{aggregate size} = \min(\text{current\_aggregate\_size}, \max(\text{prev\_aggr\_size}/2, 8))$$

if any of the following cases are present:

  - aggregate tail errors > max (2 \* head aggregate errors, 4)
  - instantaneous error rate >= 50% and aggregate size > 8

## Implementation

The current implementation of the above is scattered across multiple files. It includes two steps, first collecting the statistics from the current completed aggregate and second applying the algorithm in rate probing time.

For every aggregate completed `ath_tx_num_bad_frames()` would return the number of tail errors and the number head errors in a 16-bit integer. The lower 8 bits of this would have the number of tail errors and upper byte would have number head errors. Refer to the following check in `ath_edma_tasklet()` for AR938x/AR939x chips. Also, aggregate rate control is updated through the `ath_rate_tx_complete_11n()` function.

```
ath_edma_tasklet()
{
/* rest of the code */
If (!is_ampdu) {
    /* legacy/non-ampdu frame processing */
} else {
    nbad = ath_tx_num_badfrms(sc, bf, &ts, txok);
#if ATH_SUPPORT_VOWEXT
if(ATH_IS_VOWEXT_AGGRSIZE_ENABLED(sc) ||
ATH_IS_VOWEXT_RCA_ENABLED(sc))
{
    n_tail_fail = (nbad & 0xFF);
    n_head_fail = ((nbad >> 8) & 0xFF);
    nbad = ((nbad >> 16) & 0xFF);
}
#endif
/* rest of the code */
#endif
/* rest of the code */
#endif
*/
```

```

        if (!wbuf_is_sa_train_packet(bf->bf_mpdu))
    {
        ath_rate_tx_complete_11n(sc,
                                an,
                                &ts,
                                bf->bf_rcs,
                                TID_TO_WME_AC(bf->bf_tidno),
                                bf->bf_nframes,
                                nbad, n_head_fail, n_tail_fail,
                                ath_tx_get_rts_retrylimit(sc, txq));
    }
#else
    if (!wbuf_is_sa_train_packet(bf->bf_mpdu))
    {
        ath_rate_tx_complete_11n(sc,
                                an,
                                &ts,
                                bf->bf_rcs,
                                TID_TO_WME_AC(bf->bf_tidno),
                                bf->bf_nframes,
                                nbad,
                                ath_tx_get_rts_retrylimit(sc, txq));
    }
#endif
}

```

In the function `rcUpdate_11nVivo/rcUpdate`, the number of head and tails failures would be copied to the per node `pRateCtrl` structure.

Based on the above feedback, the size of the next aggregate is chosen for a particular node. The following code segment in `ath_lookup_rate()` in the file `ath_xmit_ht.c` would make the aggregate size and throttling decision.

```

#if ATH_SUPPORT_VOWEXT
/* RCA */
if ( (ATH_IS_VOWEXT_RCA_ENABLED(sc)) &&
((WME_AC_VI == ac) || (WME_AC_VO == ac))){
    aggr_limit = MIN(aggr_limit, ( (pRc->aggrLimit)*MPDU_LENGTH ) );
}
else if ( ATH_IS_VOWEXT_AGGRSIZE_ENABLED(sc) ) {
    n_head_fail = pRc->nHeadFail ? (pRc->nHeadFail) : 1;
    n_tail_fail = pRc->nTailFail;
    n_aggr_size = pRc->nAggrSize;

    if(an->throttle) {
        an->throttle--;
    }
    if(((n_tail_fail >= MAX((sc->agthresh * n_head_fail)>>2,4)) ||
       ((n_head_fail+n_tail_fail)*2 >= n_aggr_size)) && n_aggr_size >= 8) {
        if((n_aggr_size>>1)) {
            aggr_limit = MIN(aggr_limit, MAX((n_aggr_size >> 1) *
(4 + 1540),sc->agtB_blim));
        }
        an->throttle = 10;
    }
}

```

```
#endif
```

The code also checks and changes the aggregate size based on the time the hardware took to complete the frame. The TSF is tracked for this purpose. Refer to the following code segment in the file `ath_xmit_ht.c/ath_tx_form_aggr()`.

```
#if ATH_SUPPORT_VOWEXT
    if ((ATH_IS_VOWEXT_RCA_ENABLED(sc)) || (ATH_IS_VOWEXT_AGGRSIZE_ENABLED(sc)))
) {
    if (!ATH_IS_VOWEXT_RCA_ENABLED(sc)) {
        lapsed = (currts >= firstxmitts) ? (currts - firstxmitts) :
            ((0xffffffff - firstxmitts) + currts);
        if(lapsed >= sc->agtb_tlim) {
            an->throttle = 10;
        }
    }
    if (an->throttle ) {
        an->min_depth = 1;
        bf->bf_rcs[0].tries = bf->bf_rcs[1].tries = bf->bf_rcs[2].tries
= bf->bf_rcs[3].tries = 1;
    } else {

```

## Configuration

- Compile-time configuration

Enable `ATH_SUPPORT_VOWEXT` in `build/scripts/<board>/config.<board>` for getting this feature enabled.

- Run-time configuration

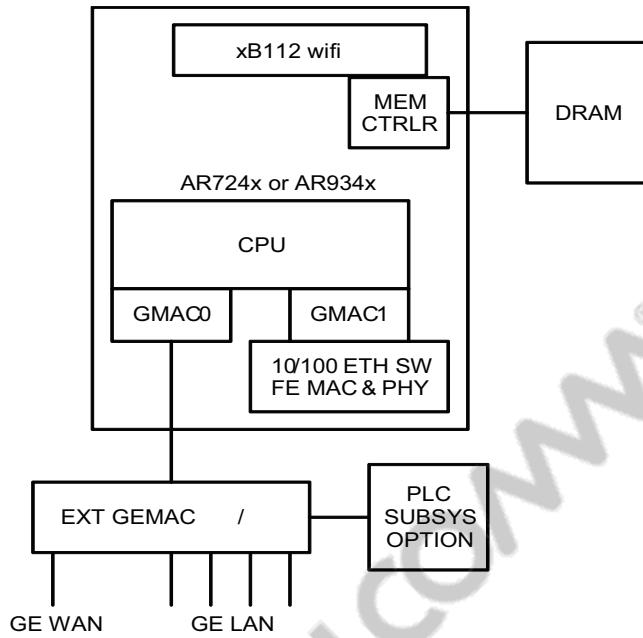
Check the current configuration of the VoW. If bit-3 is ‘1’ it is enabled already. To disable this place ‘0’ in bit-3, to enable place ‘1’.

For example, if the current configuration returned is 1, using the following command would enable this algorithm.

```
iwpriv wifiX setVowExt 0x4
```

### 18.3.2.4 Flow control

In a typical VoW AP both the wireless and Ethernet links are bridged, and traffic from the Ethernet interface is switched onto the wireless interface and vice versa. The traffic could be either destined to the wireless station that is connected to the AP or another Ethernet station that is connected to the switch. The Ethernet MAC on the chip is connected to a 4/5 port Ethernet switch similar to that shown in [Figure 18-13](#). On AR724x PB92 board GE0 is connected to S16 5 port switch and all ports are connected to single GE0. There is no WAN port.



**Figure 18-13 Ethernet Switch Example**

Because Ethernet operates at 1 Gbps and the wireless network at a much lower physical rate, there would always be speed mismatch and there would be an occasion where the wireless driver would run out buffers/descriptors. The wireless driver can run out of descriptors/buffers for two reasons.

- *Short bursts of Ethernet traffic.* Usually the application traffic from the connected machines could come in bursts. When a small/long burst of Ethernet traffic gets switched into a wireless driver, all the existing buffers would be consumed by the burst, and frequently the driver gets into no-descriptor status.
- *When the wireless traffic gets to a bad state,* a greater number of frames gets retried. As all retried frames consume transmit descriptors, and the incoming traffic would get dropped with error ‘no-descriptor’ to send.

In both the above cases the incoming traffic would get lost. This traffic can be protected (not fully, but to an extent) by employing the following solutions together.

- First, instead of dropping the frames at the wireless driver ingress, utilize the kernel network packet schedulers queuing capability. This would not completely reduce the packet loss, but would reduce it to an extent.
- Second, enable the Ethernet protocol specified PAUSE capability to pause the traffic bursting port to hold for a while, so that the wireless driver can complete the unfinished frames.

Applying both methods reduces the packet loss and improves the throughput.

The remainder of this section discusses the existing driver and current implantation of the flow control. A section on the flowmac module that does the coordination job is included.

### 18.3.2.5 Existing Implementation

- No flow control on ingress traffic

In the existing implementation (without any flow control), any traffic coming from Ethernet port is either switched on to another Ethernet port or to wireless driver based on the destination MAC address. Both the Ethernet and wireless drivers work independent of each other by forwarding frames from and to the bridge in which both the interfaces are participating.

When a quick burst of traffic is received by the GE0 on the Ethernet driver, all the frames are forwarded to the wireless driver at the same rate as they arrive. The amount of traffic switched into wireless interface depends on the number of available buffers at wireless interface. If the wireless driver has no transmit descriptor, the Ethernet frame would be dropped without informing the other components on the path. If the data frame is an UDP frame, the frame would be lost forever, causing video frame loss.

- Prioritization of traffic within the Ethernet and wireless driver

- Traffic prioritization at Ethernet driver

The existing implementation of the Ethernet driver does not impose any limits on different class of traffics it receives. The Ethernet driver considers all the traffic classes to be equal while receiving. All the traffic from the Ethernet driver would be switched to the wireless interface regardless of the traffic class. For VoW AP, it is important the video traffic be switched more often than the other classes of services.

- Traffic prioritization at wireless driver

The current wireless driver has built-in traffic discrimination at different levels. Key for the traffic prioritization at the wireless driver is either the VLAN priority or the IP packet's DSCP code. Prioritization happens at two places, first at software through the VoW transmit buffer enhancement algorithm, and second through the VSP algorithm, and third at the wireless hardware through WMM parameters.

With all the above prioritizations, the wireless driver still experiences **no buffer/no descriptor** conditions, when Ethernet traffic comes in bursts, eventually leading to real video packet loss.

If an AP has mixed ingress traffic and the AP experiences wireless congestion, it is likely that *lower priority traffic within the wireless driver can consume more number of buffers and could cause high priority traffic (video) to lose*. This condition is solved through the feature ATH\_TX\_BUF\_FLOWCTRL to some extent, but not completely.

### 18.3.2.6 Flow control – Implementation

The previous section explains the problem of flow control. The current implementation of flow control tries to address the first problem identified in previous section. The priority issue is addressed partially but cannot be solved completely. Different alternate solutions are possible to solve the priority<sup>1</sup> issue

Both Ethernet and wireless drivers have the following issues:

- First, both do not take advantage of the Linux kernel queuing capability. Both drivers are configured with a 1000 txqueue length. This means that during driver congestion, the kernel network stack can hold up to 1000 socket buffers (sk\_buff)<sup>2</sup>.

---

1. On Linux platforms, one of the proven methods is to use traffic shaping tools like tc. Though there are more than one queue disciplines possible, tests are showing HTB as more appropriate for Video AP.

- Second, the Ethernet driver and wireless driver work independently of each other. This makes it difficult for one driver to know the congested state of the other driver, resulting in frame drop/loss.

At high ingress traffic, because of above two issues and slow egress link with wireless speeds, the wireless driver frequently gets to a congested state. At this state, the wireless driver will not have a transmit descriptor (here after called no-buffer-state), because the frame would be dropped at ingress to the wireless driver. The first problem can be solved by detecting the no-buffer-condition and enabling the backlog queue for the device that is experiencing the congestion. *This implementation of flow control does this.*

Second, a separate module (`flowmac`) is implemented, to *coordinate the events between both the wireless and Ethernet drivers*. On egress device's request, `flowmac` would pause the ingress device. For example, a wireless device can generate a pause request to stop the traffic from Ethernet. This can only work with if the all the drivers are on same processor.

Even though a backlog queue can reduce the packet loss, backlog queues can overflow in the presence of long bursts of Ethernet traffic. At the time of backlog queue overflow, the Ethernet source is stalled by generating the PAUSE frame on wire<sup>1</sup>. This helps link partner to pause its traffic for some duration. Eventually traffic would resume on Ethernet when PAUSE is removed.

### **Pausing Ethernet traffic**

PAUSE on Ethernet can be generated in two ways.

- Frame a PAUSE in software and generate it out on the link
- Enable hardware flow control on both MAC and port, and stall the receive for a while, causing MAC's internal buffer overflow, that can result in PAUSE.

The second approach is chosen because *AR724x* Ethernet switch MAC drops the incoming PAUSE from the GE0/CPU MAC port.

When Rx is stalled on GE0 MAC, the first GE0's internal memory gets filled (fill thresholds can be configured) resulting in a PAUSE to switch MAC. When the switch receives the PAUSE from GE0, it would filter out the PAUSE frame. This actually results in each of the switch port's memory (thresholds are configurable) to fill up, resulting in a PAUSE on wire.

The above approach has a disadvantage. When GE0's MAC generates a PAUSE to the switch, all of the switch ports would get stopped. This causes entire switch to PAUSE traffic. So, even traffic would not flow between two switch ports. This is one of the reasons to choose a timer-based RESUME in the `flowmac` driver. This can be resolved, if there are two CPU MACS, one connecting to WAN port and other to switch.

---

2. Also called as backlog queue

1. Link partner UDP's transmit buffer would overflow when Ethernet PAUSES are held for long time. That shifts the frame drop-condition to link partner. This can be solved partially through adjusting UDP transmit buffer, but not completely.

## Enabling/Disabling the Linux Kernel Device Backlog Queue

Linux device backlog can be enabled by stopping the device through `netif_stop_queue()` and disabled through `netif_wake_queue()`<sup>1</sup>.

The current implementation is wrapped in a OS-specific function. Below is a code snippet from `osif_umac.c`.

```
#if ATH_SUPPORT_FLOWMAC_MODULE
static void
osif_pause_queue (os_if_t osif, int pause, unsigned int pctl)
{
    struct net_device *dev = OSIF_TO_NETDEV(osif);
    struct ath_softc_net80211 *scn = ath_netdev_priv(((osif_dev*)osif)->os_comdev);
    /* if pause, call the pause other wise wake */

    if (!dev || !scn) return ;

    if (pause) {
        netif_stop_queue(dev);

        if (scn->sc_ops->flowmac_pause && (pctl & 0x2)) {
            scn->sc_ops->flowmac_pause(scn->sc_dev, 1);
    }
    } else {
        netif_wake_queue(dev);

        if (scn->sc_ops->flowmac_pause && (pctl & 0x2)) {
            scn->sc_ops->flowmac_pause(scn->sc_dev, 0);
        }
    }
}

#endif
```

### Detecting pause condition

The Need-to-Pause state is triggered when a frame must be dropped due to a no-buffer condition. In this implementation, need-to-pause is triggered only when there is at least one video frame to be dropped because of a no-buffer condition. Need-to-Pause is not triggered as long as other traffic classes are dropped because of a no-buffer condition.

Theoretically, the device backlog queue can be enabled every time a no-buffer condition is seen. In practice, it reverses the priority of the video frames. Because the kernel backlog queue is a single queue, all the ingress traffic would be fed into a single queue<sup>2</sup>. This increases the probability of

- 
1. Once stalled, it is driver's responsibility to wake the queue, otherwise queues can get stuck permanently. Please refer to Linux documentation for further description.
  2. Post 2.6.15 Linux Kernels implement multiple transmit queues. Current Qualcomm drivers are not optimally ported to post 2.6.15 kernels. That imposes few limitations on flow control implementation.

video frame loss due to a drop at the tail of the backlog queue. The current implementation tries to protect video traffic.

### Current 9.2.0.x driver implementation of PAUSE

The current 9.2.0.x wireless driver is implemented in two layers; namely, LMAC and UMAC. The LMAC driver has one interface registered with the Linux Kernel (wifi0) and the UMAC registers (ath0). The current implementation tries to use the backlog queues of both interfaces.

At a Need-to-Pause condition (for video only), the first backlog queue of the wifi0 interface is enabled. Assuming both wifi0 and ath0 have 1000 buffers of backlog queue each, the first wifi0 backlog queue is enabled. When the wifi0 queue is full<sup>1</sup>, the ath0 backlog queue is enabled. Simultaneously, Ethernet pause is enabled through the flowmac module interface. See the flowmac implementation and its interfaces for further detail.

The following code excerpt is from the function `ath_tx_start_dma` in the `ath_xmit.c` file where the pause within the LMAC layer occurs.

```
/*
 * The function that actually starts the DMA.
 * It will either be called by the wbuf_map() function,
 * or called in a different thread if asynchronous DMA
 * mapping is used (NDIS 6.0).
 */
int
ath_tx_start_dma(wbuf_t wbuf, sg_t *sg, u_int32_t n_sg, void *arg)
{
    ...
    if (txctl->ismcast) {
        /*
         * When servicing one or more stations in power-save mode (or)
         * if there is some mcast data waiting on mcast queue
         * (to prevent out of order delivery of mcast,bcast packets)
         * multicast frames must be buffered until after the beacon.
    */
        if (txctl->ps || avp->av_mcastq.axq_depth) {
            send_to_cabq = 1;
#if ATH_TX_BUF_FLOW_CNTL
            buf_used = &sc->sc_cabq->axq_num_buf_used;
#endif
        }
    }

#if ATH_TX_BUF_FLOW_CNTL
    ...
    /*
     * This the using of tx_buf flow control for different priority
     * queue. It is critical for WMM. Without this flow control,
     * at lease for Linux and Apple OS, WMM will fail even HW WMM queue
     * works properly. Also the sc_txbuf_free counter must be count
     * precisely, otherwise, tx_buf leak may happen or this flow control

```

1. Backlog queue is considered full when 872 of 1000 buffers are already queued.

```

        * may not work.
        */
if (unlikely((buf_used > MIN_BUF_RESV) &&
             (sc->sc_txbuf_free < txq->axq_minfree)))
{
    . . .
}

#if ATH_SUPPORT_FLOWMAC_MODULE
    /* let us get the flowcontrol support for only video traffic,
     * rest of the traffic would continue to work in old way. If
     * there is a mixed traffic, and there is no buffer for Video,
     * rest of the traffic also would stall.
     *
     * This path is covered if
     * - VSP is disabled and Video traffic is coming through here
     * - Irrespective of VSP and traffic is not Video traffic
     */
if (!sc->sc_osnetif_flowcntrl || (txctl->qnum != WME_AC_VI)) {
    sc->sc_stats.ast_tx_nobuf++;
    sc->sc_stats.ast_txq_nobuf[txctl->qnum]++;
    atomic_dec(&an->an_active_tx_cnt);
    return -ENOMEM;
} else {
    /* inform kernel to stop sending the frames down to ath
     * layer and try to send this frame alone.
     */
    if (sc->sc_osnetif_flowcntrl) {
        ath_netif_stop_queue(sc);
    }
}
#else
    sc->sc_stats.ast_tx_nobuf++;
    sc->sc_stats.ast_txq_nobuf[txctl->qnum]++;
    atomic_dec(&an->an_active_tx_cnt);
    return -ENOMEM;
#endif
}
. . .
#endif
. . .

/* For each sglist entry, allocate an ath_buf for DMA */
TAILQ_INIT(&bf_head);
for (i = 0; i < n_sg; i++, sg++) {
    int more_maps;
    ath_get_buf_status_t retval;

    more_maps = (n_sg - i) > 1;

#endif ATH_TX_BUF_FLOW_CNTL
. . .
    if(ATH_IS_VOWEXT_FAIRQUEUE_ENABLED(sc)) {

```

```

ATH_TXBUF_LOCK(sc);

/* do not stall the queues here, Video still can run as
 * when frames from BE/BK are de-queued.
 */
if (ath_deq_single_buf(sc,
                        &sc->sc_txq[sc->sc_haltype2q[HAL_WME_AC_BK]]))
{
    /* no buf is available from BK ac, so try BE ac */
    if (ath_deq_single_buf(sc,
                           &sc->sc_txq[sc->sc_haltype2q[HAL_WME_AC_BE]]))
    {
        /* no buf is available from BE ac also, so try to
         * get buf from vi stream
        */
        ATH_TXBUF_UNLOCK(sc);
        retval = ath_tx_get_vibuf(sc, sg, &bf, &bf_head,
                                  txctl, txq, buf_used);
        stall_queue = 1;
    }
    else
    {
        /* successfully dequeud one buf so get buf from buf
         * pool
        */
        sc->vsp_bkpendrop++;
        ATH_TXBUF_UNLOCK(sc);
        retval = ath_tx_get_buf(sc, sg, &bf, &bf_head,
                               txctl->qnum, buf_used);
        * If both present, WMM and aggregate scheduling
        * shall give some fairness to Video.
        */
        stall_queue = 0;
    }
}
else
{
    /* successfully dequeud one buf so get buf from buf
     * pool
    */
    sc->vsp_bependrop++;
    ATH_TXBUF_UNLOCK(sc);
    retval = ath_tx_get_buf(sc, sg, &bf, &bf_head,
                           txctl->qnum, buf_used);
    /* see above comment */
    stall_queue = 0;
}
else
{
    retval = ath_tx_get_vibuf(sc, sg, &bf, &bf_head, txctl,
                              txq, buf_used);
    stall_queue = 1;
}

#endif ATH_SUPPORT_FLOWMAC_MODULE
/*

```

```

        * Either the fair queue is not enabled OR
        * If enabled, there is a case of video trying to grab the
        * buffers of video. Stall the queues and enable the queueing
        * for the traffic.
        *
        */
/* This path is reached only when the traffic is video traffic
 * and there is video contention */
if (stall_queue && sc->sc_osnetif_flowcntrl) {
    ath_netif_stop_queue(sc);
}
#endif
}
#else
. . .
}

```

**NOTE** The code excerpt that is shown in previous section explains how traffic is paused at UMAC layer

### Current driver implementation of RESUME operation

Both the Ethernet driver and wireless driver should not hold the PAUSE state for a long time. Resume operation is done in both in the wireless driver and Ethernet driver together. The driver should resume the traffic more frequently to empty the buffers frequently. Each of the driver implements its own approach to resume the traffic.

Current implementation resumes the traffic in two ways

- Interrupt driven – All the frames that are transmitted in wireless driver show up in the transmit complete path, which is through the `transmit` tasklet of the wireless driver. In an 802.11n driver, typically one aggregate is transmitted and one aggregate is completed in the transmit complete path.

Both the `wifi0` and `ath0` interface transmit scheduling is re-enabled when an aggregate/frame is successfully completed in the transmit tasklet. This approach is taken by the wireless driver.

Refer to the function `ath_tx_tasklet` in `ath_xmit.c` at L layer.

```

/*
 * Deferred processing of transmit interrupt.
 */
void
ath_tx_tasklet(ath_dev_t dev)
{
. . .
#if ATH_SUPPORT_FLOWMAC_MODULE
    if (sc->sc_osnetif_flowcntrl) {
        ath_netif_wake_queue(sc);
    }
#endif
}

```

The following code shows how it is done at the UMAC layer; all successful frames are marked as `IEEE80211_TX_FLOWMAC_DONE`. In `ieee80211_complete_wbuf()` in `ieee80211_wireles.c`, queues are awakened.

```

/* if_ath.c */
static void
ath_net80211_tx_complete(wbuf_t wbuf, ieee80211_tx_status_t *tx_status)
{
    struct ieee80211_tx_status ts;

    ts.ts_flags =
        ((tx_status->flags & ATH_TX_ERROR) ? IEEE80211_TX_ERROR : 0) |
        ((tx_status->flags & ATH_TX_XRETRY) ? IEEE80211_TX_XRETRY : 0);
    ts.ts_retries = tx_status->retries;

    . . .
    ath_update_txbf_tx_status(ts, tx_status);
#endif ATH_SUPPORT_FLOWMAC_MODULE
    ts.ts_flowmac_flags |= IEEE80211_TX_FLOWMAC_DONE;
#endif
. . .

    ieee80211_complete_wbuf(wbuf, &ts);
}
/* ieee80211_output.c */
void
ieee80211_complete_wbuf(wbuf_t wbuf, struct ieee80211_tx_status *ts )
{
    . . .

#if ATH_SUPPORT_FLOWMAC_MODULE
    /* if this frame is getting completed due to lack of resources, do not
     * try to wake the queue again.
     */
    if ( (ts->ts_flowmac_flags & IEEE80211_TX_FLOWMAC_DONE)
        && ni->ni_vap->iv_dev_stopped
        && ni->ni_vap->iv_flowmac) {
        if (ni->ni_vap->iv_evtable->wlan_pause_queue) {
            ni->ni_vap->iv_evtable->wlan_pause_queue(
                ni->ni_vap->iv_ifp, 0, ni->ni_vap->iv_flowmac);
            ni->ni_vap->iv_dev_stopped = 0;
        }
    }
#endif
    ieee80211_free_node(ni);
}

```

- Timer based – If Ethernet driver is paused for more duration than configured threshold (for example, 128 ms), traffic would RESUME at Ethernet automatically. The reason for this approach is driven by the current hardware implementation of the switch. Please refer to limitations section for more details. Flowmac module does this for Ethernet devices automatically. Every time the traffic is paused, a timer is started for configured duration. If the traffic do not resume within this configured duration, traffic would resume.

```

int
flowmac_pause(void *handle, int pause, u_int32_t duration)
{
    if (!handle) return -EINVAL;

    if (check_pause_limits(duration)) {

```

```

    . . .
    tfmac->current_flow_state = pause;
    /* start the timer if going to PAUSE state */
    if (pause && duration) {
        mod_timer (&tfmac->resume_timer,
                   jiffies + msecs_to_jiffies(duration));
        flowdev->stats.stalls++;
    }
    . . .
} else return -EINVAL;
return 0;
}
/* timer function */
void
flowmac_resume_timer(unsigned long cnxt)
{
    flow_mac_t *tfmac = (flow_mac_t *)cnxt;
    if (!tfmac) return;

    /* resume the device without any regard */
    tfmac->flowdev->stats.forced_wakeup++;
    flowmac_pause(tfmac, 0, 0);
}

```

### 18.3.2.7 Flowmac driver module

As mentioned in previous sections, the current implementation of the flow control tries to protect frame loss at wireless interface due to short/long Ethernet bursts. Also as mentioned, both the wireless driver and Ethernet driver work independent of each other. Unless, they communicate each other, it is not possible to pause the source of the traffic.

In current implementation of the flow control a small Linux kernel module (flowmac) is implemented to communicate the pause/resume status between the wireless and the Ethernet drivers. The following are the requirements considered for implementation.

- Flow control module coordinates between multiple devices
- Flow control module ignores device-specific PAUSE and RESUME semantics and how it is implemented. Different drivers can chose to implement it their own way.
- Flow control module implements failover mechanisms such as not keeping the devices in PAUSE for long times, to protect traffic across the ports.
- Flow control recognizes different priorities of frames; however rate limiting these priorities is still in the scope of the respective drivers
- Rate control is not in the scope of flow control. It can only transparently communicate between the driver modules involved, but should not act on those events.

With above requirements, flowmac implements the following APIs. Rate limiting low priority traffic and recovery mechanism are not implemented in the current implementation. This supports PAUSE/RESUME only.

## Registering, un-registering, pausing and resuming (API)

All drivers that require flow control should register with the flow control module. The flow control module maintains the list of devices in a simple linked list with all the options that are requested by the driver when the driver is initialized.

```
void *flowmac_register(struct net_device *dev, flow_ops_t *ops,
                      int ing_flow_ctl, int eg_flow_cntrl, int ingrate, int egrate);
```

dev – Pointer device specific `net_device` structure. The current code does not look into any of the fields in this structure. It still considers it as an opaque pointer. The flow control (flowmac) module requires this to pass down any of the `ioctls` needed, or to stop and start the device if necessary.

`flow_ops_t` – As mentioned above, pausing and resuming the traffic is in the scope of the device. Every device implements its own way of pausing and resuming. Every device should register function pointers that would implement the following:

- Pause              implements both pause/1 and resume/0
- Rate\_limit        processes any rate limit commands
- Notify\_events    passes any notifications from flow control module to the driver; for example, forced resume

```
typedef struct _flow_ops {
    u_int8_t (*pause)(struct net_device *dev,
u_int8_t pause, u_int32_t duration);
    u_int8_t (*rate_limit)(struct net_device *dev,
flow_dir_t dir, void *rate);
    int (*notify_event)(struct net_device *dev,
flowmac_event_t event, void *event_data);
} flow_ops_t;

int ing_flow_ctl
int eg_flow_cntrl
```

The device is ingress and egress flow controllable. Currently there is no specific use case for egress flow control. It is possible that applications above the drivers (such as `ioctls`) can request to change from auto rate to manual rate.

```
int flowmac_unregister(void *handle);
```

Unregister a device to remove from flow control

```
int flowmac_pause(void *handle, int pause, u_int32_t duration);
```

When one particular device detects a resource condition to pause its data provides, it needs to call this function with the handle provided.

```
int get_pause_state(void *handle);
```

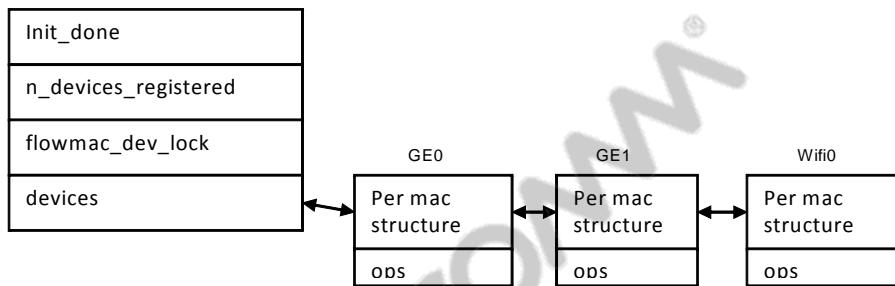
Interface to query if a driver is in paused state. Device is not queried for any status. It is from the internal state of the flow control

```
int flowmac_rate_limit(void *handle, flow_dir_t dir, void *rate);
```

If any device wanted its traffic to be rate limited, it can call this function. A corresponding call would be made into all the drivers that are ingress rate limited. The rate structure is understood only by the driver, not by the flow control module.

## Implementation

The current implementation tries to translate all the flow control events from the one driver into the other driver. Every time a new driver is registered a data structure for the device is created with all the parameters that are passed through registration. A linked list (of type list\_head) is maintained for all the devices that are registered. [Figure 18-14](#) shows the data structures involved in the module.



**Figure 18-14 Flow Control Data Structures**

### Configuration and run time information

- Compile time configuration

Current implementation of the code can be found at the linux/drivers/net/flowmac path of the current drivers.

Linux/drivers/net/flowmac/flowmac\_api.h – Flowmac API file

Linux/drivers/net/flowmac/flowmac.h – Flowmac internal definitions

Linux/drivers/net/flowmac/flowmac.c – flowmac implementation

- Build related files

Linux/kernels/mips-linux-2.6.31/arch/mips/configs/ap113\_defconfig

Linux/kernels/mips-linux-2.6.31/drivers/net/Kconfig

- CONFIG\_FLOWMAC - To compile the flowmac module, kernel module flag CONFIG\_FLOWMAC should be defined in the kernel configuration file. This enables the flowmac code in both the Ethernet driver and in the flowmac driver.

- ATH\_SUPPORT\_FLOWMAC - To enable the flowmac support in the current wireless driver, enable the above flag in the following file.

build/scripts/board\_type/config.board\_type

The above two flags are dependent on each other. Make sure that either both are ON or both are OFF.

### Run time configuration

The current release drivers has this compiled in by default for the select boards. For example. on AP113 and PB9x-2.6.31 builds it is always compiled. But this is disabled at run time. As this feature requires one more driver support, most of the run time configurations need to be selected at

module load time or bring up time. If any run time configuration need to be passed down to all modules as module parameter, all the drivers need to be reloaded after enabling the flow control. That is similar to rebooting the AP. There is no immediate plan to get complete run time configuration.

To enable flow control, the following variables need to be added to the configuration space (use `cfg -a` followed by `cfg -c`)

- `LAN_WLAN_FLOWCONTROL` – enables the flow control between LAN and WLAN drivers. Set this ‘3’ to enable this and ‘0’ to disable.
- `ETH_FLOW_CONTROL0` – Set this to enable flow control on Ethernet interface – 0
- `ETH_FLOW_CONTROL1` – Set this to enable flow control on Ethernet interface – 1

On Ethernet it is possible to enable/disable Rx only flow control or Tx only flow control, or both. Below are the list of configurations available for variables `ETH_FLOW_CONTROLx`.

- |   |                                   |
|---|-----------------------------------|
| 0 | Board-specific default            |
| 1 | Allow only Tx Flow control        |
| 2 | Allow only Rx Flow control        |
| 3 | Allow both Tx and Rx flow control |
| 4 | Force ‘No flow control’           |

### Run time statistics

There are not many statistics added to the flowmac driver. Both the wireless and flowmac driver have few statistics.

- Wireless driver statistics – Athstats has two variables: `netif_stop` and `netif_wake` to count the number of times the wifi0 device is paused and awakened.
- The flowmac driver has similar statistics to know how many times Ethernet driver is stopped and resumed. This can be obtained from the file `/proc/flowmac/stats`.
- This lists the number of devices that are registered with flowmac driver. Also it provides the number times the Ethernet PAUSE is enabled and disabled on the Ethernet device.

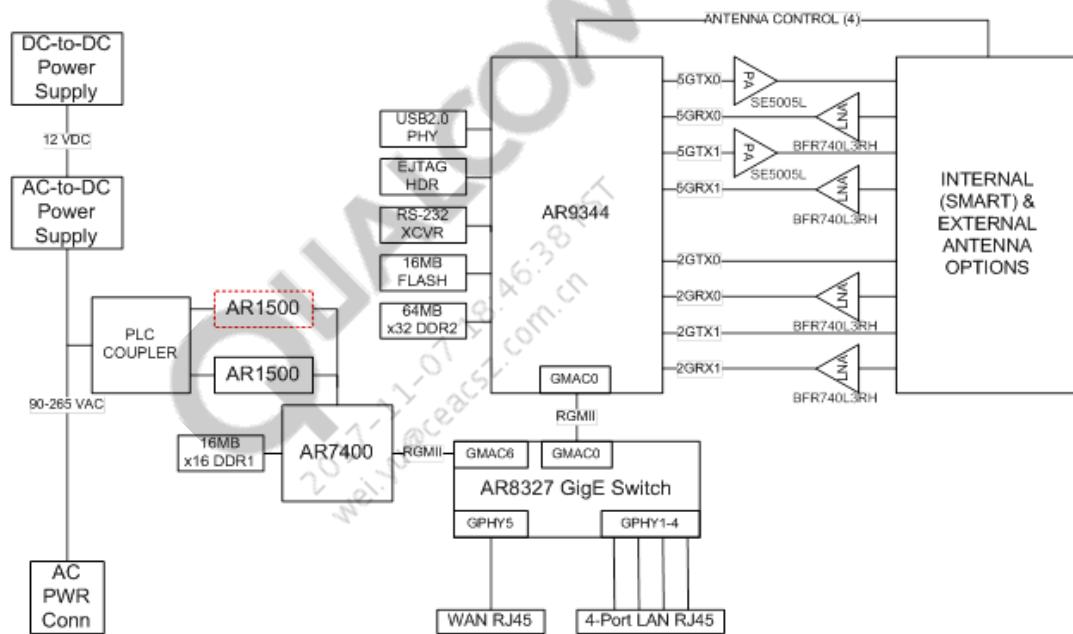
### Limitations

There are occasions where the PAUSE can be held for a long duration. Flow control should recover from this type of error. Starting a timer for the duration required. If the RESUME is not happening within that duration, traffic need to be resumed.

**Theoretically this is possible, but this is likely caused by another hardware or software issue. To catch this, this is intentionally left not implemented.**

- Traffic into wireless/PLC device could be either from the GE0 or GE1. The flowmac driver does not try to identify from where the traffic is originating. User interaction or intrusion into the bridge interface is required to detect and pair the interfaces that can request and act to pause events.

- As mentioned in previous sections, if Ethernet is paused through flowmac module, it is likely that traffic from one port to the other ports within the switch could get lost. This is one of the limitations of the current hardware design.
- Rate limiting the traffic at ingress is possible, but the measure of rate is not clear.
- Operating system portability is not considered
- Although different priorities in Ethernet can be used, the method of keeping these priorities common across the drivers is yet to be identified.
- This module is not recommended for the APH126, which runs PLC firmware on a different board that has its own processor, while wireless runs on a AR934x/AR935x board with MIPS processor. Unless there is some hardware interaction between PLC and AR934x/AR935x this cannot be utilized. See [Figure 18-15](#) for the APH126 platform block diagram.



**Figure 18-15 APH126 Reference Design Block Diagram**

Also, the AR7400 and AR9344 are connected through AR8327. If any of the ports on AR8327 pauses, communication between AR7400 and AR934x/AR935x would completely fail.

- The flow control (flowmac) module does not try to implement end-to-end flow control. This is implemented in the best interest of the wireless driver to reduce the packet loss when there is burst of Ethernet traffic, not vice versa. For example, if there is a 100 Mbps Ethernet link as egress and 350 Mbps of wireless interface as ingress traffic, flowmac does not flow control the traffic. One of the reasons for this is that the wireless traffic does not have any interoperable protocol specified for the PAUSE/RESUME mechanism. Any solution would become potentially inoperable across multiple implementations. Because of this, the current implementation becomes one-way (protect wireless from Ethernet) flow control

### 18.3.2.8 Rate Control and Aggregation Sizing Algorithm for Video (RCA)

The performance requirements are different for video and BE traffic. BE traffic requires high average throughput, whereas video requires low MAC level packet losses and low latency for constant bit-rate traffic. Current rate control implementation optimizes for maximum average TCP throughput at the expense of increasing packet loss for UDP. This leads to glitches and stalling in video streaming during channel outage. The objective of this methodology is to get the best possible throughput for video even during channel outage.

This algorithm also merges the current aggregation sizing algorithm with the rate control algorithm. Packet error rate depends on both aggregation size and current rate. Here, both aggregation size and rate are optimized together to get best throughput in all the channel conditions.

One part of this algorithm also works on dynamic RTS transmission. RTS transmission increases overhead if used in a clean channel, whereas it reduces collision if used in an environment with hidden nodes and interference. This algorithm dynamically enables and disables RTS transmission.

#### Theory of operation

The legacy algorithm monitors average PER and selects rate to optimize the throughput. PER of every packet is computed by dividing number of sub-frames failed in a packet to the total number of sub-frames transmitted in that packet. A running average of this PER is maintained for each rate. The legacy algorithm uses this average PER and computes the effective throughput (also known as “goodput”) for every rate. Goodput is defined as ‘rate in kbps\*(1-PER)’. It then chooses a rate which has maximum goodput.

Video optimized rate control algorithm uses both average and instantaneous PER to choose rate and aggregation size. Instantaneous PER helps in fast channel tracking and thus, ensures low PLR and optimized throughput even during temporary channel outage. When the channel is good and consistent, the algorithm uses average PER to compute the rate and maximizes throughput. But when the channel is not-good and varying, the algorithm also uses instantaneous PER and chooses a rate which reflects the current channel condition. This change is temporary. When channel recovers, algorithm again uses average PER to compute final rate. It also maintains a rate versus aggregation size table for each node. It stores the maximum aggregation size to be used with each rate.

The performance of the rate-control algorithm depends a lot on the PER estimate. This algorithm also improves the PER computation methodology. PER depends on both rate and aggregation size. Higher /lower aggregation size can artificially increase/decrease the PER estimate.

This algorithm also extends the concept of a hardware retry series to software. A hardware retry series immediately reduces the rate for the failed packet and retransmits it. If the transmission is successful (received a block ack), but has some sub-frames in error, then those sub-frames are retransmitted by software at the same rate. This increases the latency and also the probability of losing the packet by excessive retries. Theoretically, if the PER for a rate is 20%, then the probability of losing a frame by excessive retry itself is around 1E-7. This algorithm reduces the rate of the packet, if the last packet is transmitted with many sub-frames in error. It helps in reducing the worst case latency.

### 18.3.2.9 Implementation

The algorithm implementation resides mainly in two files: ratectrl\_11nViVo.c and ath\_xmit\_ht.c. ratectrl\_11nViVo.c contains all the rate control functions required by video and voice traffic. Since this algorithm merges aggregation size algorithm to the rate control, all the aggregation sizing code which is related to video has been moved to ratectrl\_11nViVo.c.

The RCA code will execute only if ATH\_SUPPORT\_VOWEXT is defined and the RCA enable bit is enabled through iwpriv. ATH\_IS\_VOWEXT\_RCA\_ENABLED is the macro used in the code, which checks for the RCA enable flag.

The following are the implementation details.

- Controlling per packet aggregation size
  - Maintaining rate vs. aggregation size table to indicate maximum allowed aggregation size for each rate. For this an array has been added: maxAggrSize in the TX\_RATE\_CTRL structure. This array is maintained for each node.
  - The max aggr size in the table is always  $\geq 8$  and  $\leq 32$
  - Ageing the table periodically: maxAggrSize table is incremented every sc→rca\_aggr\_uptime milliseconds by a factor called sc→aggr\_aging\_factor. Both the parameters ‘rca\_aggr\_uptime’ and ‘aggr\_aging\_factor’ are programmable. These parameters should be modified only by the developer during debugging.
- Using Instantaneous PER
 

Defining three ‘bad PER conditions’ to detect sudden channel variations based on the transmit status of the last aggregate.

  - Hardware excessive retry (provided RTS–CTS was enabled for that packet)
  - When the PER of last packet is greater than 50%.
  - When the goodput of the first half of last aggregate is greater than the goodput of the last aggregate. This indicates high PER due to longer aggregate
- Whenever the above bad PER conditions are noticed, change the rate and aggregation size in the following way:
  - After hardware excessive retry or if the last packet’s PER  $> 50\%$ 
    - For the immediate recovery, use the next lower supported rate for the next transmission (provided this rate is above the minimum rate threshold for video).
    - Reduce the max allowed aggregation size for this rate by 12.5%. This number is programmable and named ‘excretory\_aggr\_red\_factor’ in the code. This will ensure lower aggregation size for the same rate, if the algorithm chooses the same rate again for the next packet.
  - If the Goodput of the full packet is less than the goodput of the first half of the packet:
    - Reduce aggregation limit for the next packet by half.
    - Penalize the maximum aggregation size for this rate by ‘badper\_aggr\_red\_factor’. This is programmable.

Where goodput of the first half =  $(1 - \text{inst PER of fast half of the packet}) * \text{best throughput with the current rate and half the aggr size}$ .

Goodput of the entire aggregate = (1 - inst PER of the packet)\*best throughput with the current rate and current aggr size.

The above equation has been simplified in the code to reduce complexity.

- After hitting the above conditions, don't increase the rate for a fixed duration (10 ms).
- In addition to this, whenever a bad channel condition is hit, disable the following for a fixed number of packets. (This is tracked by a variable named throttle. Whenever the above conditions are true, it is set to 10 and is reduced by 1 for every other packet.)
  - Disable hardware retries, only if the rate is above a minimum rate threshold
  - Disable double buffering of packets in hardware queue. This is done so that the rate control feedback can be given to the immediate next packet.
  - Enable RTS – CTS.
- In the normal channel condition (when the poor channel conditions are not true):
  - choose the rate by using the basic rate control algorithm, which optimizes goodput
  - gradually increase the aggregation size:

If the chosen rate is higher than the last rate, limit the aggregation size of the current packet to a fixed number.

Gradually, increase the aggregation size in steps of 8 until it reaches a maximum aggregation size for that rate.

The algorithm also includes some PER estimation enhancements. The following are some of these enhancements:

- When the transmitter does not get a block ack for the packet, then the hardware itself retransmits the same packet. This is known as a hardware retry. A hardware retry does not mean 100% error. It can be result of a collision or some block ack bug. So, the PER need not be updated for every transmission failure. This is handled in the following way:
  - If the packet is successful after more than 1 hardware tries (at the same rate), don't update the PER for the failed attempts.
  - If the packet fails for all the hardware retries, then check the max aggregation size for that rate. If the max aggregation size is below a threshold (named as 'sc->per\_aggr\_thresh' in the code), then only update the PER. Max aggregation size is decremented with every excessive retry. The threshold is set such that the PER will be updated after around three such hardware retries. 'per\_aggr\_thresh' is programmable and can be controlled through iwpriv commands. It should be modified only by developer during debugging.
- PER depends a lot on aggregation size:
  - PER seen for a single subframe packet, can corrupt the PER estimate of an aggregate.
  - Legacy rate control algorithm periodically sends probe packets to check the performance with higher rates. These packets are transmitted without any aggregation. If a probe packet (with 1 subframe) succeeds, then don't update the PER for this attempt. If a packet with 1 subframe succeeds, then the current PER for this packet is 0%. This doesn't mean that the higher aggregates will also succeed with zero PER. It can artificially reduce the PER.
  - However, if a single subframe packet fails, then it indicates that the PER is bad for this rate and the average PER can be updated with the PER of this attempt.

## Configuration

- Compile-time configuration

Enable ATH\_SUPPORT\_VOWEXT in build/scripts/<board>/config.<board> for getting this feature enabled.

- Run-time configuration

  - setVowExt

Check the current configuration of the VoW. If bit-5 is ‘1’ it is enabled already. To disable this, place ‘0’ in bit-5, to enable place ‘1’. For example, the feature is enabled by using the following command:

```
iwpriv wifiX setVowExt 0x10
```

  - set\_rcaparam

Note that this command is available for developer debugging only. Some of the thresholds can also be configured by using the set\_rcaparam iwpriv command:

```
iwpriv wifiX set_rcaparam <value>
```

The argument of the command <value> should be chosen such that:

- Interval to increase MaxAggrSize for all the rates (ms): sc→rca\_aggr\_uptime = <value> & 0x000000FF
- MaxAggrSize threshold to enable PER computation in case of excessive retries: sc→per\_aggr\_thresh = (<value> & 0x0000FF00) >> 8;
- MaxAggrSize table periodic increment factor (#DataFrames): sc→aggr\_aging\_factor = (<value> & 0x000F0000) >> 16;
- MaxAggrSize Reduction factor (after excessive retry): sc→excretry\_aggr\_red\_factor = (<value> & 0x00F00000)>>20; (reduces max aggr size by 1/(2^excretry\_aggr\_red\_factor)\*100 percent)
- Aggregation size penalty if PER increases (directly reduce the MaxAggrSize by this number): sc->badper\_aggr\_red\_factor = ((<value> & 0x0F000000)>>24);

All the above parameters have been optimized in the algorithm and should be changed only by developer during debugging. These parameters are dependent on each other and should be changed together.

### 18.3.2.10 Video Stream Protection (VSP)

Video streaming has stringent QoS requirements in terms of PLR (packet loss rate) and jitter/latency. These stringent QoS requirements mandate dedicated wireless bandwidth to support video applications.

The dynamic nature of the shared wireless channel poses challenges in providing dedicated wireless bandwidth to video streams. Bandwidth may not be guaranteed at all times. Even though the wireless network is able to support multiple video streams initially, changing medium conditions may create a situation where net available bandwidth is not sufficient for all incoming video streams. In this case, all the video streams get impacted and there are glitches on all video streams severely impacting the video quality and user experience.

Video stream protection tries to address this situation in a best effort manner. Instead of impacting all the video streams, it tries to penalize few streams and try to maintain the video QoS for all other video streams.

## Configuration

- Compile-time configuration

Enable ATH\_SUPPORT\_VOWEXT in build/scripts/<board>/config.<board> for getting this feature enabled. It also expects ATH\_TX\_BUF\_FLOW\_CNTL to be defined (enabled by default).

- Run-time configuration

  - (get)set\_vsp\_enable

VSP can be enabled/disabled using this iwpriv. Default value is 1 (Enabled) if VoW is compiled.

```
iwpriv wifiX set_vsp_enable 1 To enable VSP
iwpriv wifiX set_vsp_enable 0 To disable VSP
```

```
iwpriv wifiX get_vsp_enable To get the current status of VSP
```

  - (get)set\_vsp\_thresh

iwpriv command to set/get goodput threshold value, which is used by the VSP algorithm to differentiate between bad and good video streams. Default value is 4096000 (40 Mbps converted in Kbps)

```
iwpriv wifiX set_vsp_thresh value set Goodput threshold value (in kbps)
```

```
iwpriv wifiX get_vsp_thresh get current Goodput Threshold value (in kbps)
```

**Note:** This configuration parameter is not a trivial one, and is not meant for end users to configure. It should be used only by the developers.

  - (get)set\_vsp\_evaldur

iwpriv command to set/get VSP algorithm evaluation interval. This is the time after which vi streams are (re)evaluated to be bad or good. Default value is 100 ms (unit is ms)

```
iwpriv wifiX set_vsp_evaldur value set VSP Algo Eval Duration (in ms)
```

```
iwpriv wifiX get_vsp_evaldur get current value of VSP Algo Eval Duration value (in ms)
```

**NOTE** This configuration parameter is not a trivial one, and is not meant for end users to configure. It should be used only by the developers.

### 18.3.3 Video over Wireless (Offload)

The following Video over Wireless enhancements is implemented in the current release.

- Reserve dedicated descriptors for Video clients.
- Retry time reduction – aggregate size control
- Penalizing scheduling opportunity of oversubscribing client (Video stream protection)

### 18.3.3.1 Reserve dedicated descriptors for VI clients

In most cases, the packet loss is caused at the ingress of the wireless access point. The transmit path requires a descriptor to queue the buffer to the hardware. In the current implementation the number of transmit descriptors varies from board to board based on the memory available. In case of offload, memory availability on target determines the number of descriptors that can be supported in the system.

Currently, it has 1424 descriptors of which 1024 are used for BE and the rest is shared among other access categories. These descriptors would be sufficient for best effort traffic, where the packets can be dropped due to ‘no-descriptor’ case.

But for the video case, transmit path should make sure that it does not drop the video packet. To achieve this, VI nodes are assigned dedicated number of descriptors. Since VI packets are jitter sensitive, there is no need to buffer too many packets. Hence, number of descriptors assigned per VI node can be configured according to acceptable jitter. To enable this configuration, module parameter “vow\_config” is introduced to configure number of VI nodes and descriptors per node. The number of descriptors per VI node should be configured to 200 if roughly 100 ms buffering for 20 Mbps ingress traffic is required.

#### Configuration

VoW configuration involves the following:

Configuration of the number of Video (VI) links supported and number of descriptors reserved (dedicated) per VI link. The number of Video links and Tx descriptors per link can be configured by setting “vow\_config” module parameter for the ‘umac.ko’ module at insmod.

vow\_config is defined as a 32 bit unsigned value of which upper 16 bits are used to configure number of VI quality nodes required and Lower 16 bits to configure the number of descriptors reserved/dedicated for each VI node.



**Lower 16 bits:** The number of descriptors (can say buffers) reserved/dedicated for each video link. Indicates VI packet buffering capacity inside the WLAN driver (200 roughly indicates 100 ms buffering capacity for 20 Mbps incoming VI stream)

**Upper 16 bits:** Number of video links (nodes that are supported) that the AP can guarantee video quality. Beyond this, video quality is not guaranteed, video streams uses common pool of descriptors which may result in sub-optimal video performance.

#### Example

1. vow\_config=0x40190 means 4 VI nodes with 400 descriptors each. This translates to 4 VI quality nodes which can support up to 200 ms of buffering per node/stream.
2. vow\_config=0x800c8 means 8 VI nodes with 200 descriptors each. This translates to 8 VI quality nodes which can support up to 100 ms of buffering per node/stream.

3. `vow_config=0x5012C` means 5 VI nodes with 300 descriptors each. This translates to 5 VI quality nodes which can support up to 150 ms of buffering per node/stream.

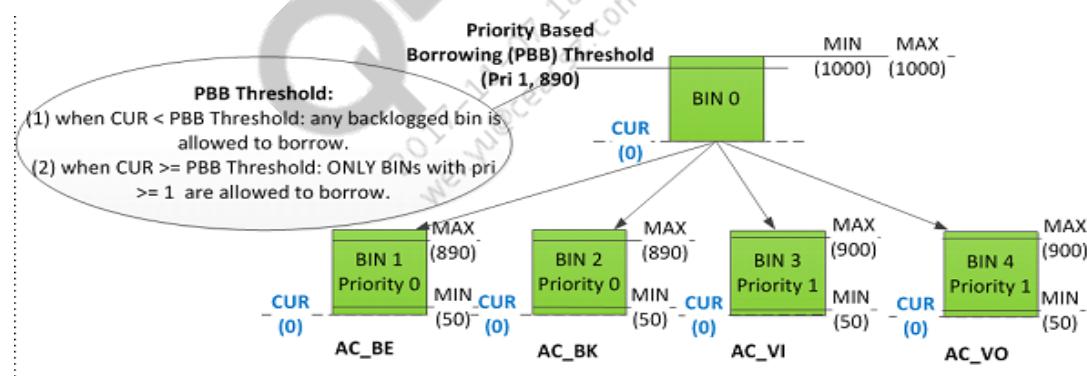
#### NOTE

- Due to memory constraints current release can reserve only 1600 descriptors for VI nodes. This means it can support up to 8 VI nodes with 200 descriptors per each VI link. Configuring higher numbers will result in failure to load umac module.
- Due to memory constraints, enabling WoW configuration automatically limits the maximum VAP or client supported on AP. With WoW enabled, AP can support minimum of 8 stations and 8 VAPs.

#### Design

In 10.2 code base, Hierarchical Credit Management (HCM) framework is used to manage descriptors allocation for different ACs. This architecture defines various bins with min and max defined such that a bin is guaranteed min number of descriptors and it can borrow up to max number of descriptors. Each AC is mapped into one bin. Also, this architecture defines Priority Based Borrowing (PBB). PBB can be set such that bins with lower priority don't borrow descriptors beyond this threshold.

Figure 18-16 depicts one example WMM configuration using HCM framework.



**Figure 18-16 WMM Configuration (HCM Framework)**

In Figure 18-16, total numbers of descriptors are 1000. BE/BK bin is guaranteed 50 descriptors and it can borrow up to 890. Similarly, VI/VO bin is guaranteed 50 and can borrow up to 900. And if PBB is set as 890 for priority 1, then only VI/VO (bin priority 1) can get descriptor if current usage is beyond 890.

#### Allocation of WoW bin

The WoW bins are allocated whenever a VI TID is added.

#### Free WoW bin

The WoW bins are freed whenever a VI TID with WoW bin is deleted.

### 18.3.3.2 Excessive retry drops

Packets that fail to get BA continuously for N times will be dropped. These drops are termed as excessive retry drop. For non-VoW clients and non-VI ACs, frames will be dropped if it fails to receive 16 consecutive BAs. Since VoW provides separate dedicated jitter buffers for VoW peers, frames are dropped due to excessive retry only if its jitter buffers are full and there are 16 consecutive BA failures. This ensures that the required latency as configured in `vow_config` is provided.

### 18.3.3.3 Aggregate sizing scaling

#### SIFS Bursting Disabled

Aggregate size scaling defines the scaling factor used to limit the txop for a particular AC. Usually, each transmission gets 4ms txop. Reducing txop in case of high contention scenario helps in Jitter. Since VI is jitter sensitive, scaling can be applied in high contention scenario. VoW provides an `iwpriv` command to enable and set aggregate Size Scaling factor. Aggregate size scaling value set is used as a scaling factor. The txop assigned to an AC is set as `default_txop >> aggregate_size_scaling` for that AC.

- For BE/BK, scaling is applied only in the presence of VI traffic.
- For VO/VI, scaling is applied if there are more than two active VI traffic.

With no VI traffic, BE/BK will take the default scheduling opportunity which is 4ms (no impact on the peak throughput numbers for BE TCP/UDP).

#### SIFS Bursting Enabled

With SIFS bursting enabled default max scheduling opportunity for all clients is 12ms burst. Aggregate size scaling value set is used as a scaling factor as below-

- For BE/BK, scaling is applied only in the presence of VI traffic.
- For VO/VI, scaling is applied if there are more than two active VI traffic.
- For scaling factor 1, each client gets 2ms burst as scheduling opportunity. For scaling factor greater than 1, each client gets 1ms burst.

#### Settings

```
iwpriv athN <AC> <RTS(1|0)> <scaling_factor> <BW>
```

RTS and BW will be ignored and it should be set to zero.

AC – can be one of the following.

0 – BE

1 – BK

2 – VI

3 – VO

Scaling\_factor - Value ranging from 0 to 3. 0 means scaling is disabled. Setting it to higher may have an adverse effect.

### **Example**

1. iwpriv athN acparams 0 0 2 0
  - a. Burst disabled - sets max aggregate size duration as 1ms for BE in presence of VI
  - b. Burst Enabled – 1ms burst opportunity for BE traffic.
2. iwpriv athN acparams 1 0 2 0
  - a. Burst Disabled - set max aggregate size duration as 1ms for BK in presence of VI.
  - b. Burst Enabled - 1ms burst opportunity for BK traffic.
3. iwpriv athN acparams 2 0 1 0
  - a. Burst Disabled -Set max aggregate size duration as 2ms for VI.
  - b. Burst Enabled – 2ms burst opportunity for VI traffic.

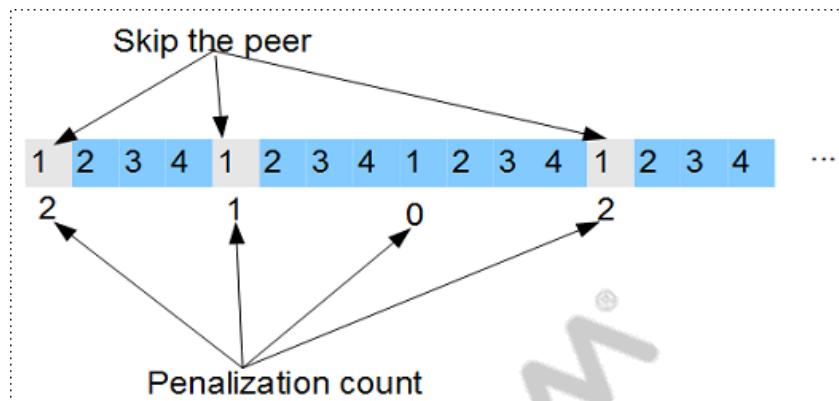
**NOTE** This is a per phy/device setting. Due to historical reason, it is using vap (ath0|1...) instead of phy (wifi0|1...) interface. Hence, this needs to be applied once for a vap belonging to a phy.

#### **18.3.3.4 Penalizing schedule opportunity / Video Stream Protection (VSP)**

In case of offload, each VI node gets dedicated descriptors. This ensures that a client with degraded medium condition uses only its quota of descriptors. However, if more clients are experiencing little bad medium condition, then this client can impact latency by affecting the retry opportunity on other streams as it will most likely occupy the medium for its full quota of txop. To minimize the affected clients, it tries to penalize few streams to maintain the video QoS for all other video streams.

### **Penalization**

Penalization is done by taking away the schedule opportunity of a penalized client. To penalize a client, current round robin scheduling is modified such that if a tid is found to be penalized, its scheduling opportunity is skipped for N iteration. Following figure depicts one such scenario. In this case, peer 1 is penalized. It loses two scheduling opportunity.



**Figure 18-17 Penalization**

#### 5.9.3.4.2 Implementation

Penalization code is implemented in `perf_pwr_offload/drivers/target/src/wlan/wal/AR/tx_sched/tx_sched.c`.

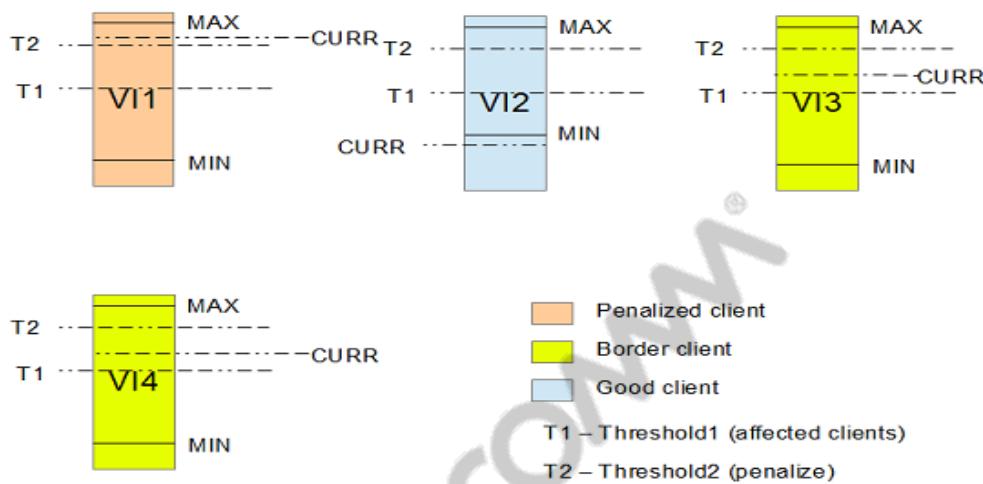
```
Function - tx_sched_algo
{
...
    if(WAL_TID_IS_PENALIZED(tid)) {
        /*This tid is penalized. Reduce it penalize count and skip this tid if
penalization count is non-zero*/
        A_UINT8 pc;
        pc = WAL_TID_GET_PENALIZE_COUNT(tid);
        if(pc) {
            pc--;
            WAL_TID_SET_PENALIZE_COUNT(tid, pc);
            /* Skip this tidq, */
            TAILQ_INSERT_TAIL(&schedule_queue, tid);
            continue;
        }
        else {
            //Reset the penalization count.
            WAL_TID_SET_PENALIZE_COUNT(tid, VOW_PENALIZED_COUNT_DEFAULT);
        }
    }
...
}
```

#### Selection of penalization client

Since all clients get same number of descriptors, client's current descriptor usage count can be used to find out a peer that is over subscribing the resources. Over subscribing could be due to degraded medium condition or increase in ingress rate. In either case, client is oversubscribing resources and considered for penalization.

Two thresholds are defined for VoW bins. Threshold t1 corresponds to usage count beyond which the stream is considered to be affected stream. Threshold t2 corresponds to usage count beyond

which the stream is considered over subscribing and a potential candidate for penalization. Figure 18-18 depicts the thresholds.



**Figure 18-18 VoW Bin Thresholds**

If a client's current usage crosses threshold t1, it is marked as affected client. If it crosses threshold t2, it is marked as penalizable client. Whenever a peer's usage crosses t2, it will be penalized if there is another client with its current usage beyond threshold t1 (i.e., affected client).

Peer is de-penalized

1. When its current usage falls below threshold t1.
2. All affected peer's usage falls below threshold t1.

Since it doesn't provide any benefit to penalize all the clients, the total number of clients penalized at any time is limited to two.

## Limitation

Current VSP implementation does not address admission control contention scenarios. Since it only looks at queue depth for penalization, there is a possibility that each client with similar bad rates can get penalized at different time. As future enhancement, VSP algorithm can be made to select same client during such contention scenarios based on AID.

Jitter buffers assigned to each client is static. It can hold packets only up to configured number of buffers. If there are huge rate fluctuations resulting in high latency, there can be packet drops. However, these drops are limited to only those clients. As a future enhancement, these buffers can be made dynamic depending on the goodput and ingress rate of client. So that unused buffers from good clients that require less descriptors can be used for other clients.

## Configuration

In current implementation there are no user configurable items for penalization.

Total number of clients to be penalized is derived from the number of VI nodes configured in `vow_config`.

Threshold t1 is set to  $\frac{1}{2}$  of number of descriptors configured per VI node. Threshold t2 is set to  $\frac{3}{4}$  of number of descriptors configured per VI node.

## 18.4 Smart Antenna

The Smart Antenna technology is a selection diversity-based technology targeted at the latest Qualcomm Technologies Wi-Fi chipsets, and is part of Video over Wireless (VoW) enhancements. Currently VoW is being used as a generic technology. A Smart Antenna system consists of internal antennas that are replacement for existing external antennas. External antennas have limitations such as space constraint, orientation sensitivity, and so on. These are addressed in a Smart Antenna system. A Smart antenna system consists of one printed and one stamp antenna connected to each Tx/Rx chain.

The Smart Antenna Algorithm selects best possible antenna combinations among different combinations of stamped and printed antennas to provide the best performance.

### 18.4.1 Smart Antenna System

The Smart Antenna algorithm is designed for Smart Antenna-enabled Qualcomm Technologies chipsets. Depending on the number of chains supported, each chain is connected to two omnidirectional antennas in which one is vertically polarized and the other is horizontally polarized.

For each chain, the software will choose the best antenna to be used for both transmission and reception. For example, the AR958x/AR959x has 3 chains, so there are 8 possible combinations of antennas to be chosen from ( $2 \times 2 \times 2 = 8$ ).

The software uses the Antenna Control fields of the transmit descriptor and the Default Antenna register to control the selection of the antennas. The Antenna Control field in the transmit descriptor controls a bit pattern that is shifted out on GPIO pins before transmission of a frame. The Default Antenna register controls a bit pattern that is shifted out on the GPIO pins after transmission of a frame, which is used for all reception until the next transmission. GPIO pins are connected directly to the switch for each chain. [Figure 18-19](#) shows the basic connection between GPIOs and antennas.

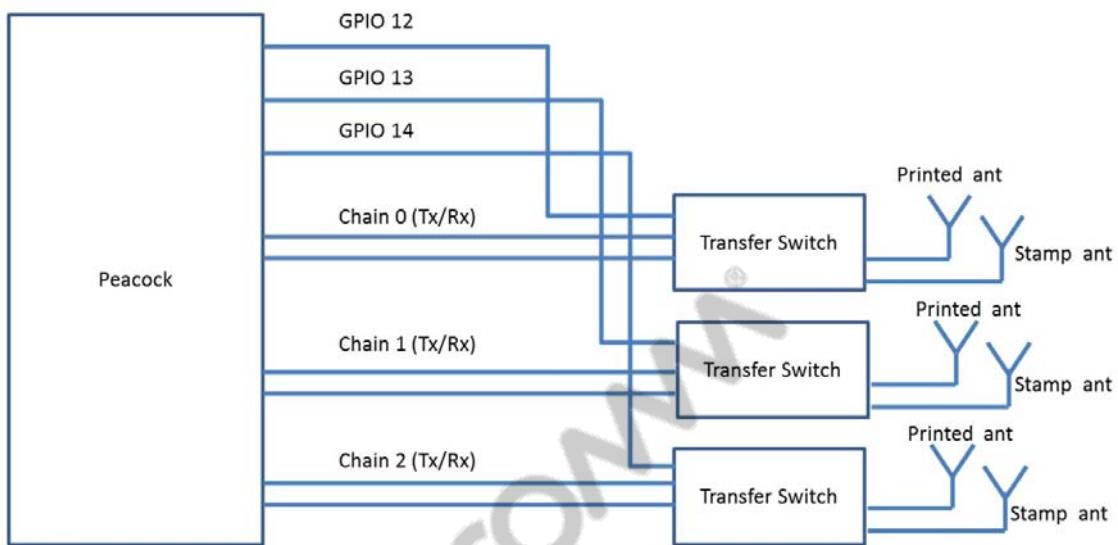


Figure 18-19 Smart Antenna Block Diagram (3x3 Chipset Shown)

2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

### 18.4.1.1 Antenna Selection parameters

The antenna combination (for either selection or omni-directional) can be chosen to minimize/maximize particular objective functions. The parameters used can be:

- PER
- RSSI
- Channel State matrix

The initial implementation will attempt to maximize an objective function that is a function of the user level throughput ( $Rate * (1 - PER)$ ) and the RSSI, similar to the objective function used for Rate Control. Future enhancements may use the Channel State matrix.

### 18.4.2 Theory of Operation

Antenna selection happens based on the stats that are collected from the training of an antenna combination. Antenna training is basically sending or receiving some packets to collect the antenna selection metrics.

Training can happen in two ways

- Using proprietary self-generated packets
- Using existing traffic

Based on the usage one of the appropriate methods will be used for training.

#### 18.4.2.1 Training sequence

##### Assumptions

- The antenna training sequence selection algorithm will run with beamforming off.
- Before the training sequence starts, the transmitter and receiver can each hear each other with whatever antennas they are using for receive and transmit.

##### Frame-based training

- In this mode the transmitter will cycle through its combinations for training without any indication to the receiver.
- Transmitter itself will collect stats. At the end of training sequence, it selects the best antenna with the collected stats.

#### 18.4.2.2 Transmit Antenna Selection

Transmit antenna selection has three steps: pre-training, training, and selection logic. Pre-training is optional and will be done based on the requirement.

## Pre-training

Training for PER/RSSI metrics involves sending a number of packets to determine proper estimates. The current algorithm starts training from the maximum rate and continues down until less than 100% PER is observed. In the worst case, all the rates are tried, which is very costly operation in terms of network bandwidth, channel utilization, and time.

To optimize training time and channel usage, pre-training is done. Pre-training aims to reduce the total number of packets and time to select antenna combination in the training process.

Pre-training is used to stabilize the rate control algorithm to the operating rate. This will be done by using either network traffic or mixed mode proprietary packet generation based on the configuration. The mixed-mode proprietary packet generation approach is same as for training which is explained later in the training section.

## Primary metric

PER stats collected from training are used as primary metric for selecting best antenna. The antenna that has the best PER will be selected as the best antenna. If the PER difference is less than a certain configured threshold (for example, -4) in a comparison between two antennas, it is treated as a conflict. A secondary metric will be applied in case of conflicts.

## Secondary metric

If there are any conflicts in the primary metric, that is, if they are almost equal, then the secondary metric will be used to find the best antenna. Block ACK RSSI is used as a secondary metric for transmit antenna selection.

## Usage for RSSI metric

As per-chain RSSIs are available, the following approach is used for comparing RSSIs:

1. Find the minimum RSSI among all the chains for each antenna combination.
2. Maximum of the minimum RSSI combination is selected as best antenna.
3. If there is a conflict in maximum of minimum RSSIs, the antenna with maximum RSSIs among other chains is selected as best antenna.

## Training with self-generated proprietary traffic

1. Do a pre-training and get the current operating rate that should be used for training.
2. Get set of rates negotiated for the node to be trained.
3. Send N number of frames of length L on each antenna combination. Send sounding packet for each antenna change.
4. Collect average PER and average RSSI values based on the ack/Back reception.
5. Repeat 3 and 4 for X number of times to average out the changing channel conditions over time.
6. Use averaged PER and RSSI values as a metric to find the best antenna.

**NOTE** Implementation details are discussed [Section 18.4.3](#).

## Training with existing traffic

Some of the ongoing network traffic packets are selected for training. VI, BE and BK traffic are used for training. Only the first attempt of transmission for VI, BE is used for training to reduce jitter. For BK, up to three software transmission attempts are used for training. In every software transmission attempt only the first rate set is used for training. Other rate sets are filled with the rates from the rate control algorithm. Only the first three rate sets are used from the rate control algorithm and are filled in the last three rate sets.

For the selected training packets, a particular rate and antenna are selected. All the antennas are cycled to get PER and RSSI of the trained packets. The rate will be selected for training in such a way that in one antenna combination it should give a PER of less than the UPPER BOUND configured threshold percent.

The current antenna selection will happen in a binary search kind of fashion, once we get the statistics (PER and RSSI) for current antenna and trained antenna. Based on the best results among two, the current antenna will be selected. This will make sure that as soon as training happens for the best antenna it will get selected.

The following is the selection algorithm based on PER:

1. Get the current operating rate using pre-training or rate control algorithm.
2. Select default antenna as training antenna.
3. Train antenna at operating rate to get stats (PER, RSSI)
4. Move rate up or down based on the values of the stats; that is, extremes are within a LOWER BOUND and UPPER BOUND range of high and low values. Otherwise store the stats and go to the next step.
5. If rate is changed, get the updated stats for that particular rate.
6. Repeat steps 4 and 5 until the process settles on a particular rate.
7. Select the next antenna as the training antenna.
8. Train the antenna at the operating rate to get stats (PER,RSSI)
9. Compare metrics and select best among previously selected antenna and trained antenna.
10. Repeat steps from 7 until we train all the antennas.
11. When all the antennas are trained, the best antenna will be selected.

Since training with network traffic has disadvantages like dependency on the network traffic, which cannot be controlled, a mix of both training with network traffic and proprietary packet generation will be used to achieve best results. This mode is called as mixed mode training.

## Training in case of insufficient traffic when existing traffic is chosen for training (mixed mode training)

When there is no traffic or if the traffic is not enough for training, proprietary traffic will be generated to make sure that training gets completed in time and the best antenna is selected for subsequent traffic.

There will be a timer maintained by the smart antenna algorithm to keep track of the training packets that are getting transmitted in the interval.

Whenever the smart antenna software algorithm finds that there is no training traffic in the last interval, it will send a small chunk of the proprietary packets for training purposes. This will happen until the training completes. These will be generated in the BK traffic category so as not to impact other traffic. Even if there is no network traffic, training is eventually completed.

All the traffic generation happens in the work queue so that it doesn't affect the existing traffic.

## Initial Training

At the time of association, initial training will happen to select the best antenna for transmission of further packets. This will happen automatically.

Training can happen either using existing traffic or through proprietary traffic.

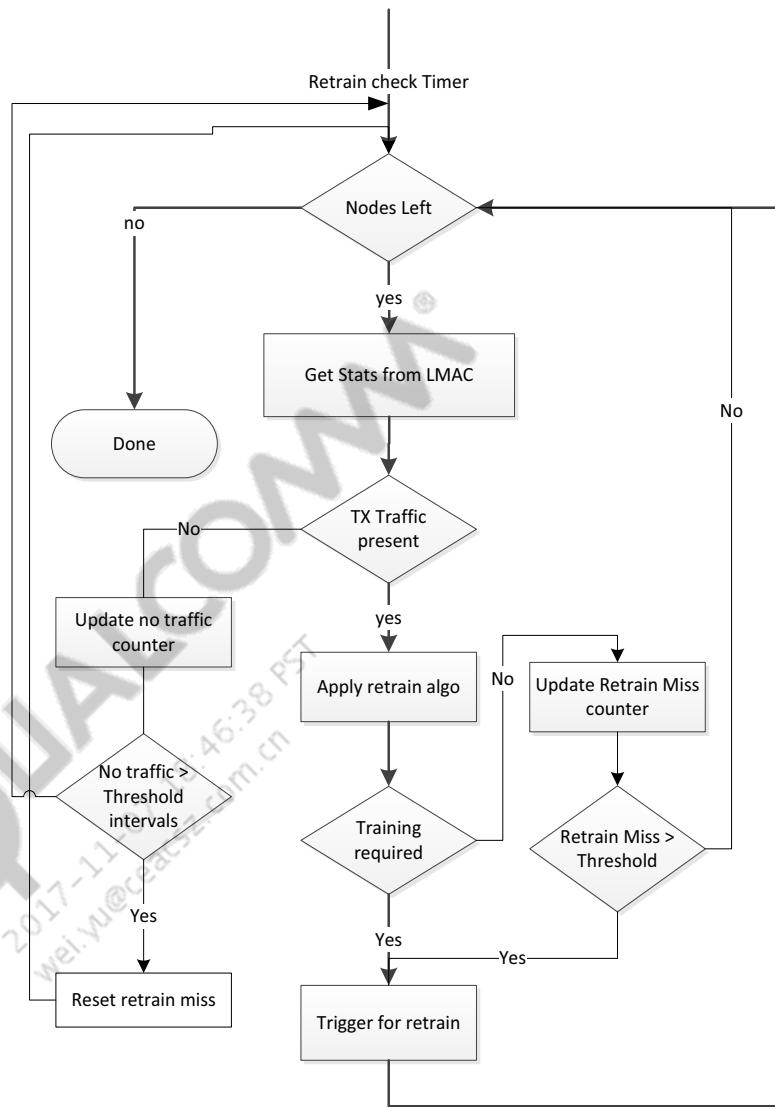
**NOTE** Initial implementation will be done with proprietary traffic and experimentally this can be changed to existing traffic based on the results.

## Dynamic Training

There is a chance that channel conditions might change after initial training. In that case some other antenna combination might provide good performance compared to the selected antenna. To address this, retraining will happen to find the best antenna combination based on a trigger. A trigger happens when there is change in the channel condition or periodically.

Once the retraining trigger happens training will happen across all the antenna combinations and the best combination will be selected. Existing traffic is used for retraining.

Smart antenna algorithm periodically monitors the traffic and finds triggers for dynamic training. See [Figure 18-20](#).

**Figure 18-20 Retraining Algorithm**

### Triggers

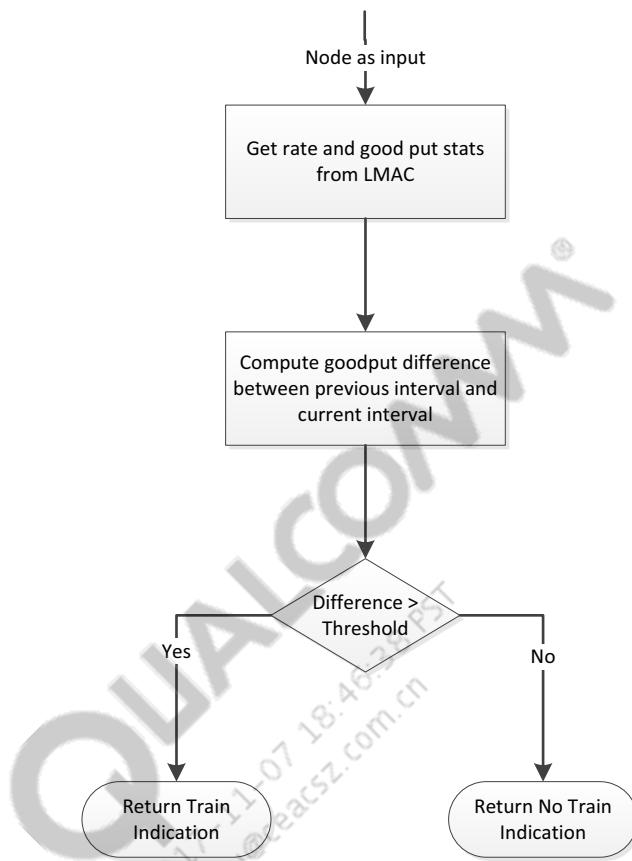
Retraining trigger happens in following cases

- Periodic: After a particular time interval lapses, retraining is triggered.
- Performance: If a transmitter detects performance has degradation or improvisation.
- Request: A receiver can request a transmitter to retrain after it changes its receive antennas. Implemented with iwpriv command.

### Algorithm for Performance based trigger

- Collect the rate stats used by a particular node periodically.
- Find the predominantly used highest rate used by the node in every retrain interval.
- If rate drop/improvement occurs select the node to retrain.

Figure 18-21 describes the performance-based training process.



**Figure 18-21 Performance-based Trigger Flowchart**

### 18.4.3 Implementation

#### 18.4.3.1 Design for using existing traffic

This algorithm is implemented in the UMAC and LMAC. The UMAC consists of control and decision logic, and is responsible for indicating to the LMAC to train a particular antenna using a particular rate. The LMAC selects some incoming traffic from the UMAC as training packets and collects PER and RSSI stats. These stats are sent to the UMAC once the training completes for that particular antenna.

The UMAC uses the PER information from the LMAC to select the best antenna. It sweeps through all the antennas and selects the best antenna.

The UMAC also monitors traffic to the LMAC. If the traffic is not sufficient (32 packets), it generates some training packets at regular intervals (500 ms interval) to make sure that training completes in time.

Figure 18-22 through Figure 18-26 indicate UMAC functionality to be done when training indication arrives.

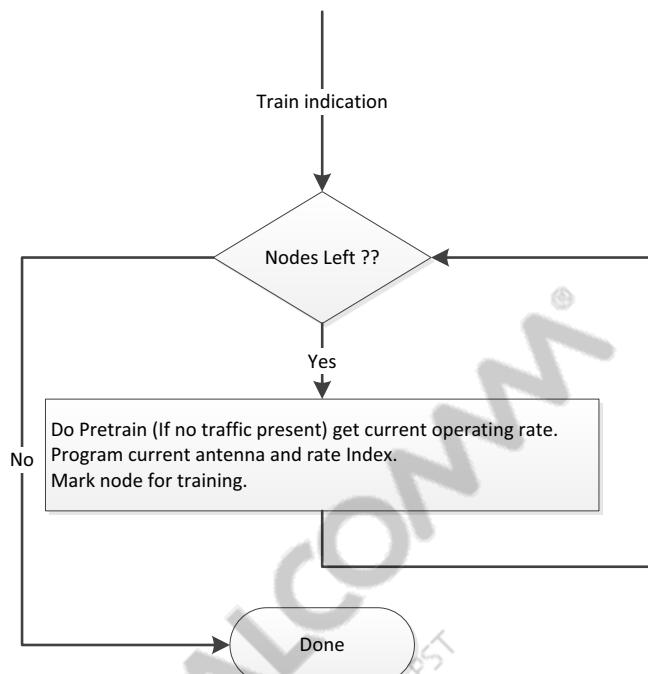
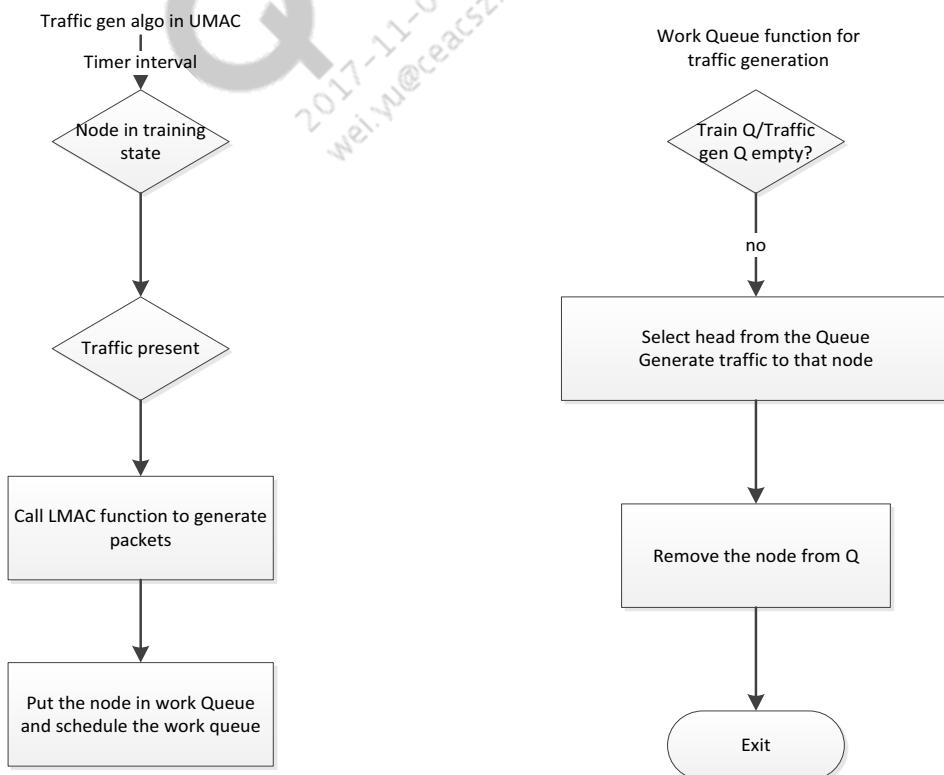
**Figure 18-22 UMAC Training Initialization Sequence****Figure 18-23 UMAC Traffic Generation (Insufficient Traffic Case)**

Figure 18-24 through Figure 18-26 describe the LMAC functionality once the UMAC has indicated training is required.

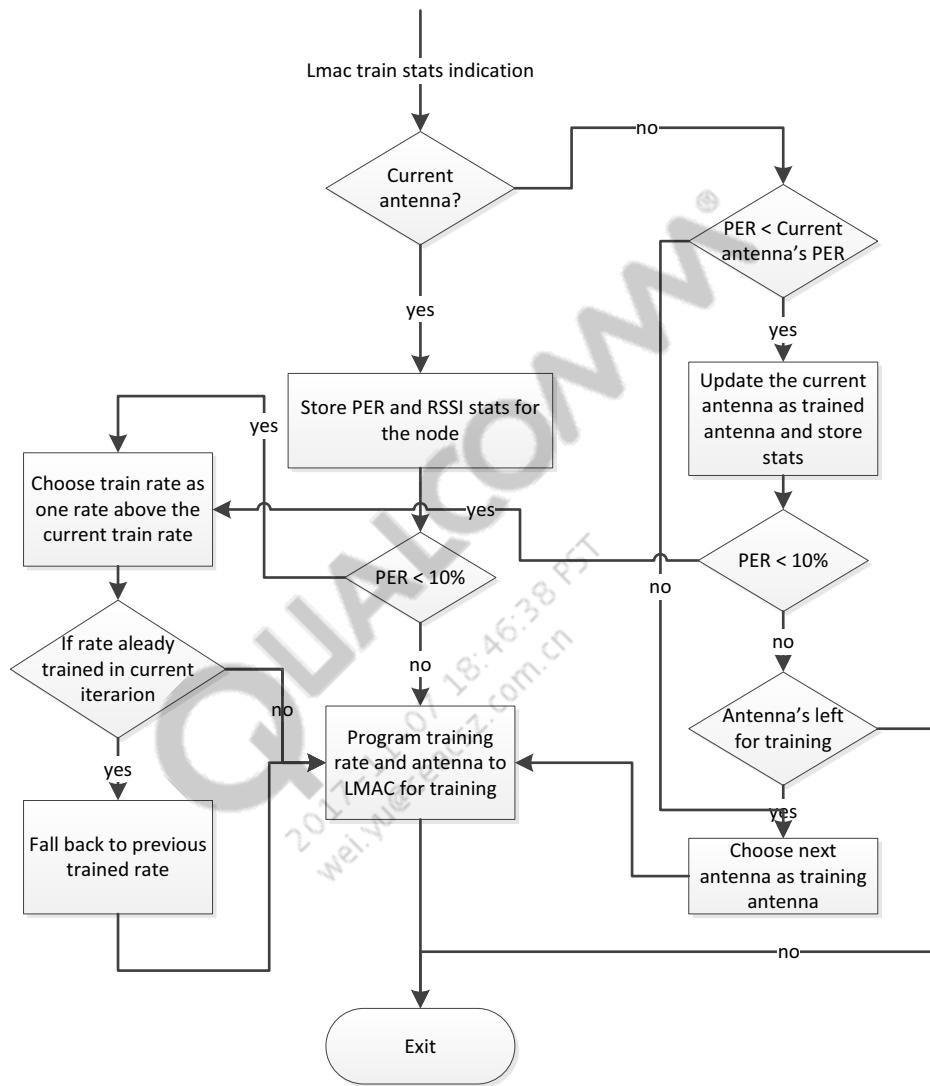


Figure 18-24 Antenna Selection when Training using Existing Traffic

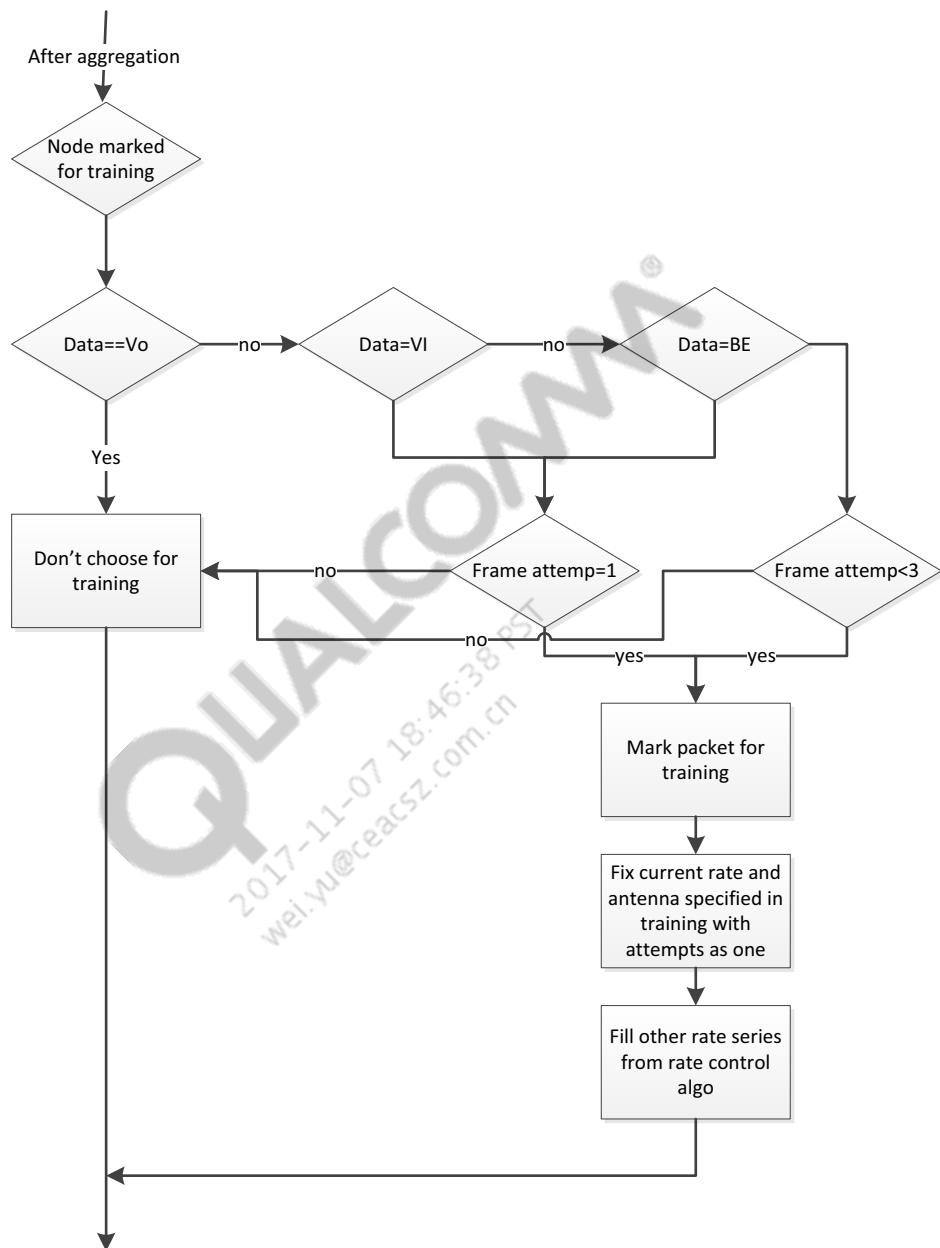


Figure 18-25 Training Packets Selection in LMAC Transmit Path

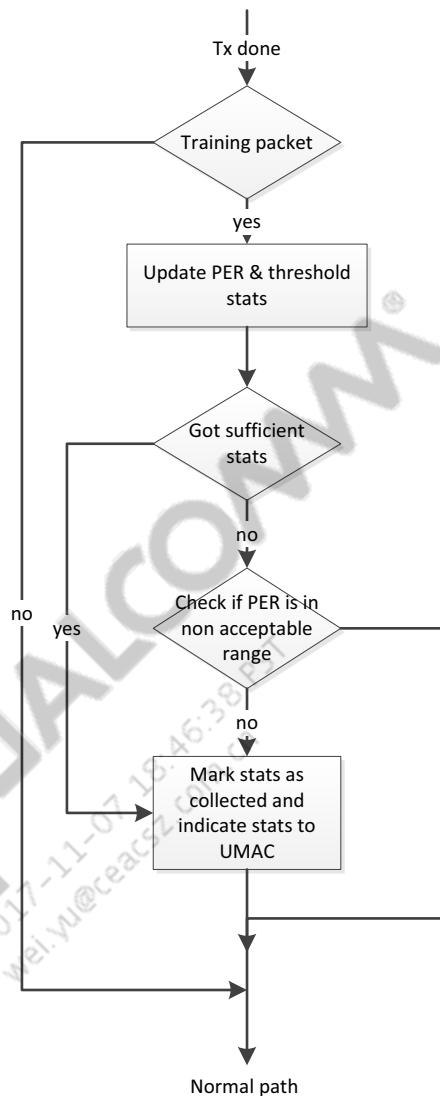


Figure 18-26 Training Stats Collection in Transmit Done Path

Table 18-3 Smart Antenna Parameter Configuration Commands

| Command                       | Format  | Description  |
|-------------------------------|---|--|
| ant_numitr                    | iwpriv athN <value>   | Value indicates hysteresis. Default Value: 3   |
| ant_retrain<br>getant_retrain | iwpriv athN <retrain interval in ms>  | Configures retrain interval in milliseconds. '0' indicates training is disabled.<br>Default Value: 1000  |
| ant_train                     | iwpriv athN ant_train <Value><br>Value - 1 will do normal training.<br>Value – RateCode (MCS value),<br>Will do fixed rate training for the specified rate. | Used to start manual TX training to all associated node in AP Mode, and TX training for associated AP in STA mode.<br>Example:<br># iwpriv ath0 ant_train 1<br># iwpriv ath0 ant_train 0x8f<br>Above command do fixrate training on MCS15. |

**Table 18-3 Smart Antenna Parameter Configuration Commands (cont.)**

| <b>Command</b>           | <b>Format</b>   | <b>Description</b>  |
|--------------------------|---|---|
| ant_trainmode            | iwpriv athN <ant_trainmode>   | '0' indicates to use only network traffic for training. '1' indicates mixed traffic<br>Default Value: 0   |
| ant_traintype            | iwpriv athN ant_traintype <val><br>Usage: value is a 32 bits, and is used to configure the following parameters.:<br>Byte3 PER lower bound<br>Byte2 PER upper bound<br>Byte1 Per diff threshold<br>Byte0 not used currently | Configures per lower bound, upper bound and diff threshold values; these values are used in antenna selection algorithm.<br>Example:<br># iwpriv ath0 ant_traintype 0x0A5A0300<br>Above command configures: PER Lower bound to 10<br>PER Upper bound to 90<br>PER Diff threshold to 3   |
| current_ant <sup>1</sup> | iwpriv athN current_ant <Value><br>Usage:<br><br><b>Byte1:</b> Association ID of STA, If Byte1 is zero configures the same antenna for all associated STAs.<br><b>Byte0:</b> Antenna value to configure for that STA.       | Sets transmit antenna combination for transmitting data packets per client.<br>This is used by developer in testing mode only.<br>Example:<br># iwpriv ath0 current_ant 0x0002<br>Configures TX antenna to 2 for all STAs<br># iwpriv ath0 current_ant 0x0102<br>Configures TX antenna to 2 for STA with AssocId 1<br># iwpriv ath0 current_ant 0x0201<br>Configures TX antenna to 1 for STA with AssocId 2<br>Default Value: 0 |
| dbgLVL                   | iwpriv athN dbgLVL 0x80   | Displays extra debug information of training process use <b>dbgLVL 0x80</b> . Used by developers in testing mode.<br>Example:<br>#iwpriv ath0 dbgLVL 0x80   |
| default_ant <sup>1</sup> | iwpriv athN default_ant <antenna_combinations><br>Deriving antenna combinations is described later in this section.   | Sets <b>Receive antenna</b> and <b>default antenna</b> which is used for transmitting Beacons and control packets.<br>Example:<br>#iwpriv ath0 default_ant 0<br>Default Value: 0  |
| get_retindrop            | iwpriv athN get_retindrop   | Displays current throughput threshold value for re-training   |
| get_retraininterval      | iwpriv athN get_retraininterval   | Displays current retrain interval value   |
| getant_numitr            | iwpriv athN getant_numitr   | Displays current hysteresis value   |

**Table 18-3 Smart Antenna Parameter Configuration Commands (cont.)**

| <b>Command</b>               | <b>Format</b>   | <b>Description</b>  |
|------------------------------|---|---|
| getcurrent_ant               | iwpriv athN <i>current_ant</i> <antenna_combinations><br><br><b>Note:</b> Make sure debug level are enabled to display the values in console. | Displays the transmit antenna combinations used by each associate node.<br><br>Example:<br><pre># iwpriv ath0 getcurrent_ant<br/>vap-0: Tx Antennas for node: 00:17:c4:81:9c:24 &gt; 0<br/>vap-0: Tx Antennas for node: 00:03:7f:34:15:09 &gt; 3<br/>vap-0: Tx Antennas for node: 00:03:7f:12:81:42 &gt; 1<br/><br/>ath0 getcurrent_ant:255<br/>Note: Ignore the value returned by application (255). This command displays the Tx antennas for all associated nodes, so it is designed to display the values in console instead of returning the value to application.</pre> |
| getdefault_ant               | iwpriv athN <i>getdefault_ant</i>   | Displays the default antenna configuration.<br><br>Example:<br><pre>#iwpriv ath0 getdefault_ant<br/>ath0 getdefault_ant:0</pre>   |
| getSmartAntennna             | iwpriv wifiN <i>getSmartAntenna</i>   | Displays the smart antenna status.<br><br>Example:<br><pre>#iwpriv wifi0 getSmartAntenna</pre>  |
| gtraffic_timer               | iwpriv athN <i>gtraffic_timer</i>   | Displays current traffic generation interval value in ms  |
| retrain_drop                 | iwpriv athN <value>   | Value indicates the threshold that is to be used to check variation in throughput for re-train trigger check<br><br>Default Value: 10   |
| retrain_interval             | iwpriv athN <value>   | Value indicates at which periodic retrain interval in minutes<br><br>Default Value: 2   |
| setSmartAntenna <sup>2</sup> | iwpriv wifiN <i>setSmartAntenna &lt;value&gt;</i>   | Bit '0' of value: Enable/Disable Smart Antenna hardware per radio.<br>Bit '1' of value: Enable/Disable Smart Antenna algorithm per radio<br><br>Example:<br><pre>#iwpriv wifi0 setSmartAntenna 3<br/>Default Value: 3<br/>Allowed values are 0, 1, 3. See Footnote 2.</pre>   |
| traffic_timer                | iwpriv athN <i>traffic_timer &lt;value&gt;</i>  | Value in milliseconds<br><br>Default Value: 100 ms on station side, 500 ms on AP side   |

1. Current antenna and default antenna should be changed only when smart antenna software algorithm is disabled, otherwise the training process may overwrite the configured values. If the smart antenna hardware is disabled, these commands will not have any effect. `iwpriv wifiN setSmartAntenna 0x1` command should be used to disable the software algorithm and keep the hardware enabled.

2. Once smart antenna is disabled by either software algorithm or both hardware and software, after enabling the smart antenna, the `iwpriv athN ant_train 1` command is required to ensure that dynamic training is functioning properly.

## 18.5 Unified Smart Antenna API

This section defines a generic API to interface third party Smart Antenna (SA) solutions with the Qualcomm Technologies Wi-Fi driver (10.x).

### 18.5.1 Glossary

| Term | Definition                    |
|------|-------------------------------|
| Tx   | Transmission                  |
| Rx   | Receive                       |
| SA   | Smart Antenna                 |
| AP   | Ieee80211 Access Point        |
| STA  | Ieee80211 Station             |
| CCP  | Configuration Control Pointer |

### 18.5.2 Objectives

APIs are defined to meet the following objectives

- Interface with 3rd party SA solutions.
- Compatible to Qualcomm Technologies legacy Smart Antenna solution.
- Common for both 11n and 11ac driver architectures.
- Host only implementation.

### 18.5.3 Smart Antenna Interface Module

The Smart Antenna Interface Module provides a uniform way to access external SA module. It contains:

- Interface functions which need to be registered by external SA module.
- Common data structures used between Wi-Fi driver and Smart Antenna module
- External SA modules must export their functions used in the Wi-Fi driver.

### 18.5.4 Application Program Interface Functions

#### 18.5.4.1 *int register\_smart\_ant\_ops(struct smartantenna\_ops \*sa\_ops)*

##### Objective

This function is called by a third party smart antenna module to register its callback methods with Qualcomm Technologies Wi-Fi driver. This function can be called from `init_module` (during `insmod`) of third party smart antenna module.

## Arguments

| Argument Name | Type                             | IN, OUT, IN/OUT | Description                                  |
|---------------|----------------------------------|-----------------|--|
| <i>sa_ops</i> | <i>struct smartantenna_ops *</i> | IN              | Pointer to <i>smartantenna_ops</i> structure |

## Return Value

Returns status of registration.

- Negative value indicates failure.
- Zero (0) indicates success.

### 18.5.4.2 *int deregister\_smart\_ant\_ops(char \*interface\_name)*

## Objective

This function is called by third party smart antenna module to deregister its callback methods with Qualcomm Technologies Wi-Fi driver for each physical radio. It can be called from *exit\_module* (*rmmmod*) of a third party smart antenna module. If a device with a given name is found, the Qualcomm Technologies Wi-Fi driver will call registered *sa\_deinit()* method and will deinitialize smart antenna for that physical radio. This call is always successful

## Arguments

| Argument Name         | Type          | IN, OUT, IN/OUT | Description                              |
|-----------------------|---------------|-----------------|--|
| <i>interface_name</i> | <i>char *</i> | IN              | Character pointer to physical radio name |

## Return Value

- Always zero (0).

### 18.5.4.3 *int sa\_init(struct sa\_config \*sa\_config, int new\_init)*

## Objective

This function initializes/re-initializes the smart antenna module per radio. This function is called after an AP or STA is initialized or whenever configuration parameters are changed.

In STA mode when operating channel is not known, smart antenna will be initialized with channel 0 (Invalid channel) to help scanning. SA module should return the best possible omnidirectional antenna when smart antenna is initialized with channel 0 (zero). Once the channel in which STA operates is known, smart antenna will be reinitialized with correct operating channel. SA module needs to return the proper Smart Antenna mode to Wi-Fi driver.

The Wi-Fi driver configures/reconfigures the smart antenna hardware (including GPIO pin and function values) based on the mode with provided configuration by SA module.

## Arguments

| Argument Name | Type               | IN, OUT, IN/OUT | Description                                       |
|---------------|--------------------|-----------------|---|
| sa_config     | struct sa_config * | IN              | Pointer to Smart Antenna configuration structure  |
| new_init      | Int                | IN              | 0 – re initialization.<br>1 – new initialization. |

## Return Value

Returns status of initialization/re-initialization.

- Negative value indicates failure.
- $\geq 0$  indicates success, GPIO configuration mode used for smart antenna control.-
  - 0 indicates GPIO serial mode
  - 1 indicates GPIO parallel mode

### 18.5.4.4 int sa\_deinit(enum radiold radio\_id)

## Objective

This function de-initializes smart antenna module.

## Arguments

| Argument Name | Type         | IN, OUT, IN/OUT | Description  |
|---------------|--------------|-----------------|--|
| radio_id      | enum radiold | IN              | Wi-Fi radio for which smart antenna de-initialization is required. |

## Return Value

Returns status of de-initialization.

- Negative value indicates failure.
- 0 indicates success.

### 18.5.4.5 int sa\_node\_connect(void \*\*ccp, struct sa\_node\_info \*node\_info)

## Objective

This function initializes/re-initializes node-specific Smart Antenna information and initializes / uses configuration control pointer (CCP) based on the following conditions:

- NULL: configuration control pointer indicates new node, SA module needs to allocate memory and initializes CCP pointer.
- Non NULL: configuration control pointer indicates re configuration. CCP is used to access the node and updates node parameters accordingly.

It is called when a new client is associated to an Access Point. The Wi-Fi driver stores CCP in each node object and uses the same CCP for subsequent calls to that link.

**NOTE** Wi-Fi driver does not check the validity of CCP pointer (if node is present Wi-Fi driver assumes it is valid CCP).

## Arguments

| Argument Name | Type                         | IN, OUT, IN/OUT | Description                            |
|---------------|------------------------------|-----------------|--|
| Ccp           | <b>void **</b>               | IN and OUT      | Pointer to CCP pointer.                |
| node_info     | <b>struct sa_node_info *</b> | IN              | Pointer to node information structure. |

## Return Value

Returns status of node connect.

- Negative value indicates failure.
- 0 indicates success.

### 18.5.4.6 *int sa\_node\_disconnect(void \*ccp)*

## Objective

This function frees node-specific Smart Antenna information maintained by a registered SA module. It must be called when a node is disconnected from the AP. The SA module needs to free CCP within this call. The Wi-Fi driver invalidates CCP for this link.

## Arguments

| Argument Name | Type          | IN, OUT, IN/OUT | Description                                  |
|---------------|---------------|-----------------|--|
| Ccp           | <b>void *</b> | IN              | Configuration control pointer for that node. |

## Return Value

- Returns status of node disconnect.
- Negative value indicates failure.
- 0 indicates success.

### 18.5.4.7 *int sa\_update\_txfeedback(void \*ccp, struct sa\_tx\_feedback \*tx\_feedback, uint8\_t \*status)*

## Objective

This function passes the transmit feedback information of a specific link to the registered SA module. The SA module will process this information. This function must be called for every

completion of a Tx packet (aggregate or individual packet – ACK/Block ACK reception or Timeout).

SA module can use this function to control and configure different smart antenna parameters. Status is used for this purpose. Based on status, the Wi-Fi driver will call registered callbacks to SA module to complete the requests.

For example, the following sequence of operations configures the Tx antenna:

1. SA module sets Status to 4 in *sa\_update\_txfeedback*.
2. Wi-Fi driver calls *sa\_get\_txantenna* to get antenna values to program.
3. SA module returns Tx antenna to Wi-Fi module.
4. Wi-Fi driver configures the Tx antenna per link.

The following sequence of operations initiates training:

1. SA module sets Status to 2 in *sa\_update\_txfeedback*.
2. Wi-Fi driver calls *sa\_get\_traininginfo* to get training parameters to program.
3. SA module returns training parameters like antenna, rates, and so on to Wi-Fi module.
4. Wi-Fi driver configures training parameters for that link and initiates training process.

**NOTE** Calling this function for each TX Packet completion adds substantial MIPS overhead, especially for 11ac throughputs.

## Arguments

| Argument Name | Type                    | IN, OUT, IN/OUT | Description   |
|---------------|-------------------------|-----------------|---|
| Ccp           | <b>void *</b>           | IN              | Configuration control pointer for that node.  |
| tx_feedback   | struct sa_tx_feedback * | IN              | Pointer to TX feedback structure.   |
| Status        | uint8_t *               | OUT             | Status information.<br>status is 8 bit bitmap value; each bit represents the following information.<br>bit 0- If set, Rx antenna configuration is required.<br>bit 1: If set, training needs to be triggered.<br>bit 2: If set, Tx antenna configuration is required (which terminates any ongoing training for that link)<br>bits 3 to 7 are not used. |

## Return Value

Returns status of Tx update feedback.

- Negative value indicates failure.
- 0 indicates success.

### **18.5.4.8 *int sa\_update\_rxfeedback(void \*ccp, struct sa\_rx\_feedback \*rx\_feedback, uint8\_t \*status)***

## Objective

This function passes RX parameters of the received packet (aggregated or individual packet) to a registered SA module. The SA module needs to process this info. This callback is called for every successfully received data packet (A-MPDU) from the client.

SA module can use this function to configure RX antenna and to initiate training. Status is used for this purpose. The Wi-Fi driver processes the request according to status information.

## Arguments

| Argument Name | Type                    | IN, OUT, IN/OUT | Description  |
|---------------|-------------------------|-----------------|--|
| Ccp           | <b>void *</b>           | IN              | Configuration control pointer for that node.   |
| rx_feedback   | struct sa_rx_feedback * | IN              | Pointer to RX feedback structure.  |
| Status        | uint8_t *               | OUT             | Status information.<br>status is 8 bit bitmap value; each bit represents the following information.<br>bit 0 – If set, Rx antenna configuration is required<br>bit 1 – If set, Training to be triggered<br>bits 2 to 7 are not used. |

## Return Value

Returns status of Rx update feedback.

- Negative value indicates failure.
- 0 indicates success.

### **18.5.4.9 *int sa\_get\_txantenna(void \*ccp, uint32\_t\* antenna\_array)***

## Objective

This function is called to get new TX antenna configuration for a specific node. It returns the antenna array (antennas for rate series) that needs to be used for all packets being transmitted to a particular node.

This function also terminates any ongoing training for that link. The Wi-Fi driver calls this callback when SA module sets bit 2 of status in *sa\_update\_txfeedback call back and clears training flag for this link*.

## Arguments

| Argument Name | Type              | IN, OUT, IN/OUT | Description   |
|---------------|-------------------|-----------------|---|
| Ccp           | <b>void *</b>     | IN              | Configuration control pointer for that node.                        |
| antenna_array | <b>uint32_t *</b> | OUT             | Array of antenna configuration values for each rate in rate series. |

## Return Value

Returns status of get TX antenna

- Negative value indicates failure.
- 0 indicates success.

### **18.5.4.10 *int sa\_get\_rxantenna(enum radioid radio\_id, uint32\_t \*rx\_antenna)***

## Objective

This function is called to get the new Rx Antenna configuration from a registered SA module. It returns the default antenna that needs to be configured for Rx.

The Wi-Fi driver calls this callback when SA module sets bit 0 of status in *sa\_update\_rxfeedback*.

## Arguments

| Argument Name | Type         | IN, OUT, IN/OUT | Description                              |
|---------------|--------------|-----------------|--|
| radio_id      | enum radioid | IN              | Wi-Fi radio identification.              |
| rx_antenna    | uint32_t *   | OUT             | Rx antenna configuration for that radio. |

## Return Value

Returns status of get RX antenna

- Negative value indicates failure.
- 0 indicates success.

### 18.5.4.11 Function name: *int sa\_get\_txdefaultantenna (enum radioid radio\_id, uint32\_t \*antenna)*

## Objective

This function is called to get the default antenna for transmission. This default antenna is used for transmission of multicast/broadcast packets (except beacon) and management packets before association.

## Arguments

| Argument Name | Type         | IN, OUT, IN/OUT | Description  |
|---------------|--------------|-----------------|--|
| radio_id      | enum radioid | IN              | Wi-Fi radio identification.  |
| Antenna       | uint32_t *   | OUT             | Default Antenna configuration for that radio – Used for multicast/broadcast packets (except for beacon) and for management packets before association. |

## Return Value

Returns status of get Tx default antenna

- Negative value indicates failure.
- 0 indicates success.

### 18.5.4.12 Function name: *int sa\_get\_bcn\_txantenna (enum radioid radio\_id, uint32\_t \*bcn\_txantenna);*

## Objective

This function is called to get the new Beacon Tx Antenna configuration for that radio. When operating in AP mode, for every beacon interval this callback is called.

When operating in STA mode, this callback is not valid. SA module needs to provide beacon antenna for requested radio.

The Wi-Fi\_33 driver programs beacon antenna for requested radio.

## Arguments

| Argument Name | Type         | IN, OUT, IN/OUT | Description   |
|---------------|--------------|-----------------|---|
| radio_id      | enum radioid | IN              | Wi-Fi radio for which beacon antenna configuration is required. |
| bcn_txantenna | uint32_t *   | OUT             | Antenna configuration for beacons.                              |

## Return Value

Returns status of get Beacon TX antenna

- Negative value indicates failure.
- 0 indicates success.

### 18.5.4.13 *int sa\_get\_traininginfo(void \*ccp, uint32\_t \*rate\_array, uint32\_t \*antenna\_array, uint32\_t \*numpkts)*

## Objective

This function is called to get training parameters for a link from registered SA module when bit 1 is set in status field returned by *sa\_update\_txfeedback()*. These training parameters will be programmed to the WLAN module.

Training is initiated through the *sa\_get\_traininginfo()* function call. Training can be stopped or terminated when one of the following occurs:

- Setting the Tx antenna; that is, setting status in Tx/Rx feedback to “Tx antenna configuration is required (bit 2)”.
- After specified numbers of training packets are transmitted.
- Setting *numpkts* to 0.

SA module needs to return proper training parameters. Configuring all training parameters is explained in the [Parameter Decoding](#) section.

The Wi-Fi driver configures training parameters for this link and initiates training by setting the training flag for this link. The Wi-Fi driver programs the training antenna and rates for TX packets for the link which has the training flag set. In Tx done path, training feedback is given to a registered SA module for each A-MPDU (that contains aggregated network packets). After transmitting the configured number of packets with training parameters, the Wi-Fi driver clears the training flag for the link.

Typical training sequence is explained in the [Parameter Decoding](#) section.

## Arguments

| Argument Name | Type              | IN, OUT, IN/OUT | Description  |
|---------------|-------------------|-----------------|--|
| Ccp           | <b>void *</b>     | IN              | Configuration control pointer for that node.   |
| rate_array    | <b>uint32_t *</b> | OUT             | Pointer to rate array.   |
| antenna_array | <b>uint32_t*</b>  | OUT             | Pointer to antenna configuration array.  |
| Numpkts       | <b>uint32_t *</b> | OUT             | Number of packets that needed to be sent with training flag marked and with provided Rates and Antenna configurations. |

## Return Value

Returns status of get train info

- Negative value indicates failure.
- 0 indicates success

### 18.5.5 Shared Data Structures

The Smart Antenna Interface Module Function Pointer Table is given below.

```
struct smartantenna_ops {
    int (*sa_init) (struct sa_config *sa_config, int new_init);
    int (*sa_deinit) (enum radioId radio_id);
    int (*sa_node_connect) (void **ccp, struct sa_node_info *node_info);
    int (*sa_node_disconnect) (void *ccp);
    int (*sa_update_txfeedback) (void *ccp, struct sa_tx_feedback *feedback,
        uint8_t *status);
    int (*sa_update_rxfeedback) (void *ccp, struct sa_rx_feedback *feedback,
        uint8_t *status);
    int (*sa_get_txantenna) (void *ccp, uint32_t *antenna_array);
    int (*sa_get_txdefaultantenna) (enum radioId radio_id, uint32_t *antenna);
    int (*sa_get_rxantenna) (enum radioId radio_id, uint32_t *rx_antenna);
    int (*sa_get_traininginfo) (void *ccp, uint32_t *rate_array, uint32_t
        *antenna_array, uint32_t *numpkts);
    int (*sa_get_bcn_txantenna) (enum radioId radio_id, uint32_t *bcn_txantenna);
};

#define SMART_ANT_MAX_RATE_SERIES 4
#define SMART_ANT_MAX_SA_CHAINS 4
#define MAX_EVM_SIZE 16 /* Max 16 pilot EVMs for 11 ac */
#define MAX_RETRIES_INDEX 8 /* For Direct attach failed tries for series0 at index
0
series1 at index 1 etc
For Offload(dyn bw) failed tries for index0 - s0_bw20,
index1 - s0_bw40 index4 - s1_bw20... index7: s1_bw160. */

#define SMART_ANT_RX_CONFIG_REQUIRED 0x01
#define SMART_ANT_TRAINING_REQUIRED 0x02
#define SMART_ANT_TX_CONFIG_REQUIRED 0x04
#define SMART_ANT_BSS_MODE_AP      0x00
#define SMART_ANT_BSS_MODE_STA     0x01
```

```

#define SMART_ANT_STATUS_SUCCESS 0
#define SMART_ANT_ANTENNA_MASK 0x00FFFFFF

#define DYNAMIC_BW_SUPPORTED 0x01

#define MAX_VALID_RATES 44 /* 32 MCS rates + 12 OFDM and CCK rates */
#define MAX_CCK_OFDM_RATES 12 /* Maximum CCK, OFDM rates supported */
#define MAX_MCS_RATES 32 /* Maximum MCS rates supported; 4 rates in each dword */
#define MAX_RATE_COUNTERS 4

enum radioId {
    wifi0 = 0,
    wifi1
};

struct sa_config {
    /* This is required in SmartAntenna at training phase which need to return
     * (antenna array and rate array) to the caller */
    uint8_t maxFallback_rates; /* 0 - single rate , 1 - 2 rates etc ... */
    enum radioId radio_id;
    uint8_t channel_num;
    uint8_t bss_mode;
};

/* TODO: ratecode_160 needs to add for future chips */
struct sa_rate_cap {
    uint8_t ratecode_legacy[MAX_CCK_OFDM_RATES]; /* Rate code array for CCK OFDM */
    uint8_t ratecode_20[MAX_MCS_RATES]; /* Rate code array for 20MHz BW */
    uint8_t ratecode_40[MAX_MCS_RATES]; /* Rate code array for 40MHz BW */
    uint8_t ratecode_80[MAX_MCS_RATES]; /* Rate code array for 80MHz BW */
    uint8_t ratecount[MAX_RATE_COUNTERS]; /* Max Rate count for each mode */
};

struct sa_node_info {
    uint8_t mac_addr[6];
    enum radioId radio_id;
    uint8_t max_ch_bw; /* 0-20 Mhz, 1-40 Mhz , 2-80 Mhz, 3-160 Mhz */
    uint8_t chainmask; /* Rx and Tx chain mask (0x23 - Tx chain mask: 3 ,Rx chain
mask: 2)*/
    uint8_t opmode; /* 0-Legacy, 1-11n Mode, 2-11ac mode */
    uint8_t channel_num; /* operating channel number */
    u_int32_t ni_cap;
    struct sa_rate_cap rate_cap;
};

struct sa_tx_feedback {
    uint16_t nPackets; /* number packets corresponding this TX feed back */
    uint16_t nBad; /* number of bad/failed packets */
    uint16_t nshort_retries[MAX_RETRIES_INDEX];
        /* For Direct attach failed tries for series0 at index 0,
           series1 at index 1 etc
           For Offload (dyn bw) failed tries for index0 - s0_bw20,
           index1 - s0_bw40, index4 - s1_bw20 ... index7: s1_bw160 */
    uint16_t nlong_retries[MAX_RETRIES_INDEX];
}

```

```

    uint32_t tx_antenna[SMART_ANT_MAX_RATE_SERIES]; /* antenna array used for
transmission */
    uint32_t rssi[SMART_ANT_MAX_SA_CHAINS]; /* rssi of ACK/ Block ACK */
    uint32_t rate_mcs[SMART_ANT_MAX_RATE_SERIES]; /* rate series used for
transmission */
    uint8_t rate_index; /* index of successful transmitted rate */
    uint8_t is_trainpkt; /* Train packet indication */
};

struct sa_rx_feedback {
    uint32_t rx_rate_mcs; /* received rate mcs */
    uint32_t rx_antenna; /* received antenna */
    uint32_t rx_evm[MAX_EVM_SIZE]; /* EVM for stream 0 , 1, 2, 3 */
    uint32_t rx_rssi[SMART_ANT_MAX_SA_CHAINS]; /* rssi of received packet */
    uint16_t npackets; /* Total number of successfully received packets of
aggregate/non-aggr */
};

```

## 18.5.6 Parameter Decoding

This section describes decoding of different parameters for specified API calls.

### 18.5.6.1 struct sa\_node\_info

| Parameter   | Possible Values  |  |
|-------------|--|--|
|             | 2.4 GHz  | 5 GHz                                  |
| mac_addr    | Mac address of associated node   | Mac address of associated node         |
| radio_id    | 0  | 1                                      |
| max_ch_bw   | 0 – HT 20<br>1 – HT 40 / VHT 40<br>2 – HT 80 / VHT 80  |  |
| Chainmask   | 0xAB; A - Chain mask for RX and B - Chain mask for TX  |  |
| Opmode      | 0 – Legacy mode (11A / 11G/ 11B)<br>1 – HT Mode (11 NA/ NG mode)<br>2 - VHT Mode (11AC Mode) |  |
| channel_num | Configured Channel Number  | Configured Channel Number              |
| ni_cap      | Node capabilities  | Node capabilities                      |
| rate_cap    | List of supported rated for that link.   | List of supported rated for that link. |

### 18.5.6.2 struct sa\_rate\_cap

Each element of the rate list array consists of rate code representing MCS, number of Spatial Streams (NSS) and Preamble.

The following is the encoding/decoding of the rate codes.

```
#define ASSEMBLE_HW_RATECODE(_rate, _nss, _pream) (((_pream) << 6) | ((-_nss) << 4)  
| (_rate))  
#define GET_HW_RATECODE_PREAM(_rcode)      (((_rcode) >> 6) & 0x3)  
#define GET_HW_RATECODE_NSS(_rcode)        (((_rcode) >> 4) & 0x3)  
#define GET_HW_RATECODE_RATE(_rcode)       (((_rcode) >> 0) & 0xF)  
  
#define IS_HW_RATECODE_CCK(_rc)(((_rc) >> 6) & 0x3) == 1)  
#define IS_CCK_SHORT_PREAM_RC(_rc)(((_rc) & 0xcc) == 0x44)  
#define IS_HW_RATECODE_HT(_rc)(((_rc) & 0xc0) == 0x80)  
#define IS_HW_RATECODE_VHT(_rc)(((_rc) & 0xc0) == 0xc0)
```

### 18.5.6.3 **struct sa\_tx\_feedback \*tx\_feedback**

**Table 18-4 Tx Feedback Decoding for 5 GHz Radio**

| Item       | Description  |
|------------|--|
| Tx Antenna | tx_antenna[0] - Holds the antenna used for Series 0 transmission<br>tx_antenna[1] - Holds the antenna uses for Series 1 transmission   |
| Data Rate  | rate_mcs[0] - 32bit value holding 4 data rates for Series 0 transmission.<br>Byte0 (LSB) holds Rate code belongs to (v)HT20 or legacy transmission<br>Byte1 holds Rate code belongs to (V)HT40transmission<br>Byte2 holds Rate code belongs to VHT80 transmission<br>Byte3 holds Rate code belongs to VHT160 transmission (for future use)<br>rate_mcs[1] - 32bit value holding 4 data rates for Series 1 transmission.<br>Byte0 (LSB) holds Rate code belongs to (v)HT20 or legacy transmission<br>Byte1 holds Rate code belongs to (V)HT40transmission<br>Byte2 holds Rate code belongs to VHT80 transmission<br>Byte3 holds Rate code belongs to VHT16<br>0 transmission (for future use)   |
| Retries    | nlong_retries[0] - Holds long retries with Series 0 BW20 (HT/VHT/Legacy) data rate (rate code belonging to Byte 0 of rate_mcs[0])<br>nlong_retries[1] - Holds long retries with Series 0 BW40 (HT/VHT) data rate (rate code belonging to Byte 1 of rate_mcs[0])<br>nlong_retries[2] - Holds long retries with Series 0 BW80 (VHT) data rate (rate code belonging to Byte 2 of rate_mcs[0])<br>nlong_retries[3] - For Future use (VHT 160MHz)<br>nlong_retries[4] - Holds long retries with Series 1 BW20 (HT/VHT/Legacy) data rate (rate code belonging to Byte 0 of rate_mcs[1])<br>nlong_retries[5] - Holds long retries with Series 1 BW40 (HT/VHT) data rate (rate code belonging to Byte 1 of rate_mcs[1])<br>nlong_retries[6] - Holds long retries with Series 1 BW80 (VHT) data rate (rate code belonging to Byte 2 of rate_mcs[1])<br>nlong_retries[7] - For Future use (VHT 160MHz)<br><br>nshort_retries[0] - Holds short retries with Series 0 BW20 (HT/VHT/Legacy) data rate (rate code belonging to Byte 0 of rate_mcs[0])<br>nshort_retries[1] - Holds short retries with Series 0 BW40 (HT/VHT) data rate (rate code belonging to Byte 1 of rate_mcs[0])<br>nshort_retries[2] - Holds short retries with Series 0 BW80 (VHT) data rate (rate code belonging to Byte 2 of rate_mcs[0])<br>nshort_retries[3] - For Future use (VHT 160MHz)<br>nshort_retries[4] - Holds short retries with Series 1 BW20 (HT/VHT/Legacy) data rate (rate code belonging to Byte 0 of rate_mcs[1])<br>nshort_retries[5] - Holds short retries with Series 1 BW40 (HT/VHT) data rate (rate code belonging to Byte 1 of rate_mcs[1])<br>nshort_retries[6] - Holds short retries with Series 1 BW80 (VHT) data rate (rate code belonging to Byte 2 of rate_mcs[1])<br>nshort_retries[7] - For Future use (VHT 160MHz) |

## RSSI

| Array Index | Bit Range | Field   | Description  |
|-------------|-----------|---------|--|
| 0           | [7:0]     | rssi[0] | RSSI of Ack on chain 0 of primary 20 MHz bandwidth. Value of 0x80 indicates invalid.   |
| 0           | [15:8]    | rssi[0] | RSSI of Ack on chain 0 of secondary 20 MHz bandwidth. Value of 0x80 indicates invalid. |
| 0           | [23:16]   | rssi[0] | RSSI of Ack on chain 0 of secondary 40 MHz bandwidth. Value of 0x80 indicates invalid. |
| 0           | [31:24]   | rssi[0] | RSSI of Ack on chain 0 of secondary 80 MHz bandwidth. Value of 0x80 indicates invalid. |
| 1           | [7:0]     | rssi[1] | RSSI of Ack on chain 1 of primary 20 MHz bandwidth. Value of 0x80 indicates invalid.   |
| 1           | [15:8]    | rssi[1] | RSSI of Ack on chain 1 of secondary 20 MHz bandwidth. Value of 0x80 indicates invalid. |
| 1           | [23:16]   | rssi[1] | RSSI of Ack on chain 1 of secondary 40 MHz bandwidth. Value of 0x80 indicates invalid. |
| 1           | [31:24]   | rssi[1] | RSSI of Ack on chain 1 of secondary 80 MHz bandwidth. Value of 0x80 indicates invalid. |
| 2           | [7:0]     | rssi[2] | RSSI of Ack on chain 2 of primary 20 MHz bandwidth. Value of 0x80 indicates invalid.   |
| 2           | [15:8]    | rssi[2] | RSSI of Ack on chain 2 of secondary 20 MHz bandwidth. Value of 0x80 indicates invalid. |
| 2           | [23:16]   | rssi[2] | RSSI of Ack on chain 2 of secondary 40 MHz bandwidth. Value of 0x80 indicates invalid. |
| 2           | [31:24]   | rssi[2] | RSSI of Ack on chain 2 of secondary 80 MHz bandwidth. Value of 0x80 indicates invalid. |
| 3           | [7:0]     | rssi[3] | RSSI of Ack on chain 3 of primary 20 MHz bandwidth. Value of 0x80 indicates invalid.   |
| 3           | [15:8]    | rssi[3] | RSSI of Ack on chain 3 of secondary 20 MHz bandwidth. Value of 0x80 indicates invalid. |
| 3           | [23:16]   | rssi[3] | RSSI of Ack on chain 3 of secondary 40 MHz bandwidth. Value of 0x80 indicates invalid. |
| 3           | [31:24]   | rssi[3] | RSSI of Ack on chain 3 of secondary 80 MHz bandwidth. Value of 0x80 indicates invalid. |

## Success Rate Index

- rate\_index: Indicates the index (series) at which data transmission got succeeded.
- "nframes == nbad" indicates excessive retry fail.

## is\_trainpkt

Flag indicating whether current Tx feedback belongs to regular packet or trained packet. 1 indicates trained packet.

#### 18.5.6.4 Tx feedback decoding for 2.4 GHz radio

| Item       | Description  |
|------------|--|
| Tx Antenna | tx_antenna[0] - Holds the antenna used for Series 0 transmission<br>tx_antenna[1] - Holds the antenna uses for Series 1 transmission<br>tx_antenna[2] - Holds the antenna uses for Series 2 transmission<br>tx_antenna[3] - Holds the antenna uses for Series 3 transmission   |
| Data Rate  | rate_mcs[0] - 32bit value holding data rate for Series 0 transmission.<br>Byte0 (LSB) holds Rate code belongs to HT20 or legacy transmission<br>Byte1 holds Rate code belongs to HT40transmission<br>rate_mcs[1] - 32bit value holding data rate for Series 1 transmission.<br>Byte0 (LSB) holds Rate code belongs to HT20 or legacy transmission<br>Byte1 holds Rate code belongs to HT40transmission<br>rate_mcs[2] - 32bit value holding data rate for Series 2 transmission.<br>Byte0 (LSB) holds Rate code belongs to HT20 or legacy transmission<br>Byte1 holds Rate code belongs to HT40transmission<br>rate_mcs[3] - 32bit value holding data rate for Series 3 transmission.<br>Byte0 (LSB) holds Rate code belongs to HT20 or legacy transmission<br>Byte1 holds Rate code belongs to HT40transmission |
| Retries    | nlong_retries[0] - Holds long retries with Series 0 for current BW<br>nlong_retries[1] - Holds long retries with Series 1 for current BW<br>nlong_retries[2] - Holds long retries with Series 2 for current BW<br>nlong_retries[3] - Holds long retries with Series 3 for current BW<br>Remaining are not used.<br>nshort_retries[0] - Holds short retries with Series 0 for current BW<br>nshort_retries[1] - Holds short retries with Series 1 for current BW<br>nshort_retries[2] - Holds short retries with Series 2 for current BW<br>nshort_retries[3] - Holds short retries with Series 3 for current BW<br>Remaining are not used.   |

#### RSSI

| Array Index | Bit Range | Field   | Description   |
|-------------|-----------|---------|---|
| 0           | [7:0]     | rssi[0] | RSSI of Ack on chain 0 of primary 20 MHz bandwidth    |
| 0           | [15:8]    | rssi[0] | RSSI of Ack on chain 0 of secondary 20 MHz bandwidth. |
| 0           | [23:16]   | rssi[0] | Not used.   |
| 0           | [31:24]   | rssi[0] | Not used.   |
| 1           | [7:0]     | rssi[1] | RSSI of Ack on chain 1 of primary 20 MHz bandwidth.   |
| 1           | [15:8]    | rssi[1] | RSSI of Ack on chain 1 of secondary 20 MHz bandwidth. |
| 1           | [23:16]   | rssi[1] | Not used.   |
| 1           | [31:24]   | rssi[1] | Not used.   |
| 2           | [7:0]     | rssi[2] | RSSI of Ack on chain 2 of primary 20 MHz bandwidth.   |

|   |         |         |   |
|---|---------|---------|---|
| 2 | [15:8]  | rssi[2] | RSSI of Ack on chain 2 of secondary 20 MHz bandwidth. |
| 2 | [23:16] | rssi[2] | Not used.   |
| 2 | [31:24] | rssi[2] | Not used.   |
| 3 | [7:0]   | rssi[3] | RSSI of Ack on chain 3 of primary 20 MHz bandwidth.   |
| 3 | [15:8]  | rssi[3] | RSSI of Ack on chain 3 of secondary 20 MHz bandwidth. |
| 3 | [23:16] | rssi[3] | Not used.   |
| 3 | [31:24] | rssi[3] | Not used.   |

### Success Rate Index

- rate\_index: Indicates the last attempted series index.
- Index indicates successful transmission index.
- If index is pointing to last index (max fall back series) then it may indicate either successful transmission at last series or excessive retry fail.
- "nframes == nbad" indicates excessive retry fail.

### is\_trainpkt

Flag indicating whether current Tx feedback belongs to regular packet or trained packet. 1 indicates trained packet.

#### 18.5.6.5 struct sa\_rx\_feedback \*rx\_feedback

rx\_rssi is same as described in txfeedback for both radios.

### Format of rx\_evm

| Array Index | Bit Range | Field      | Description   |
|-------------|-----------|------------|---|
| 0           | [31:0]    | rx_evm[0]  | EVM for pilot 0. Contain EVM for streams: 0, 1, 2 and 3.  |
| 1           | [31:0]    | rx_evm[1]  | EVM for pilot 1. Contain EVM for streams: 0, 1, 2 and 3.  |
| 2           | [31:0]    | rx_evm[2]  | EVM for pilot 2. Contain EVM for streams: 0, 1, 2 and 3.  |
| 3           | [31:0]    | rx_evm[3]  | EVM for pilot 3. Contain EVM for streams: 0, 1, 2 and 3.  |
| 4           | [31:0]    | rx_evm[4]  | EVM for pilot 4. Contain EVM for streams: 0, 1, 2 and 3.  |
| 5           | [31:0]    | rx_evm[5]  | EVM for pilot 5. Contain EVM for streams: 0, 1, 2 and 3.  |
| 6           | [31:0]    | rx_evm[6]  | EVM for pilot 6. Contain EVM for streams: 0, 1, 2 and 3.  |
| 7           | [31:0]    | rx_evm[7]  | EVM for pilot 7. Contain EVM for streams: 0, 1, 2 and 3.  |
| 8           | [31:0]    | rx_evm[8]  | EVM for pilot 8. Contain EVM for streams: 0, 1, 2 and 3.  |
| 9           | [31:0]    | rx_evm[9]  | EVM for pilot 9. Contain EVM for streams: 0, 1, 2 and 3.  |
| 10          | [31:0]    | rx_evm[10] | EVM for pilot 10. Contain EVM for streams: 0, 1, 2 and 3. |
| 11          | [31:0]    | rx_evm[11] | EVM for pilot 11. Contain EVM for streams: 0, 1, 2 and 3. |

|    |        |            |   |
|----|--------|------------|---|
| 12 | [31:0] | rx_evm[12] | EVM for pilot 12. Contain EVM for streams: 0, 1, 2 and 3. |
| 13 | [31:0] | rx_evm[13] | EVM for pilot 13. Contain EVM for streams: 0, 1, 2 and 3. |
| 14 | [31:0] | rx_evm[14] | EVM for pilot 14. Contain EVM for streams: 0, 1, 2 and 3. |
| 15 | [31:0] | rx_evm[15] | EVM for pilot 15. Contain EVM for streams: 0, 1, 2 and 3. |

### rx\_antenna

Antenna used at the time of receiving for that packet.

### Npackets

Total number of successfully received packets of aggregate/non-aggregate.

## 18.5.6.6 Training information

antenna\_array[i] represents antennas for the series i

rate\_array[i] represents rates for the series i.

- For Static BW case, rate code needs to be filled in the appropriate byte position in uint32 rate.
  - Byte 0 is used for (V)HT20/Legacy (CCK/OFDM) rates.
  - Byte 1 is for (V)HT40 rates.
  - Byte 2 is for VHT80 rates.

Wi-Fi driver relies on the position of rate code to determine the BW to be used for transmission.

Example: For HT40 transmission, rate code needs to be filled at byte 1 of uint32 rate\_array element. Similarly, for VHT80 rate code needs to be filled at Byte 2.

- For Dynamic BW case (in case SA module chooses to use this mode for training), Current BW & all subsequent lower BW rates needs to be filled in the rate\_array[i].

Example: For VHT40 dynamic mode transmission, rate\_array[i] needs to be filled with rate codes in Byte 0 & Byte 1. 11ac module supports different rates for different BWs in the dynamic BW case.

Combination of (mix) legacy and HT/VHT rates are not allowed in the rate\_array.

- Numpkts: Number of packets that needed to be sent with training flag marked with provided Rates and Antenna configurations.

## 18.5.6.7 Typical training sequence

The following operations need to be done to start and complete smart antenna training.

1. Training start is triggered by setting Bit 1 of status parameter in TX or RX feedback call back.
2. *sa\_get\_traininginfo* is called from WLAN module to get training parameters for a link from registered SA module, these parameters are programmed to a WLAN module.

3. *sa\_update\_txfeedback call back is called with Transmit parameters. Training packet is indicated by is\_trainpkt variable in Tx feedback structure.*
4. Current ongoing training termination can happen in three ways.
  - Setting the Tx Antenna; that is, setting bit 2 of status in Tx feedback (setting bit 2 in status means tx\_antenna configuration is required).
  - After specified numbers of training packets are transmitted.
  - Setting numpkts to 0.
5. To start new training with different train parameters, start from step 1; that is, set Bit 1 in TX/RX feedback.

Configuring training parameters is described in [Section 18.5.6.6](#).

#### 18.5.6.8 Sequence of operations during channel change indication to SA module

1. Wi-Fi driver calls SA module init call back *sa\_init* with new channel information (new\_init is 0). *SA module configures RX antenna and Beacon antenna based on the new channel information.*
2. *Wi-Fi module calls respective call backs sa\_get\_rxantenna and sa\_get\_bcn\_txantenna to get RX and Beacon antenna values. Wi-Fi driver programs these values to hardware.*
3. Wi-Fi driver calls SA module call back *sa\_node\_connect* with new channel information for all associated STAs (epp is not NULL; that is, reconfiguring node parameters).
4. SA module updates each node capabilities.

# 19 Wi-Fi SON Features

---

This chapter describes the Wi-Fi Self Organizing Network (SON) capabilities, such as Range Extender Placement and Auto-configuration (repacd), Daisy Chain and Best Uplink Selection, Multi-SSID and Traffic Separation, and the Powerline Communications (PLC) integration with Wi-Fi SON.

**NOTE** Starting with the QCA\_Networking\_2017\_SPF.5.x release, Wi-Fi SON support is extended to QCA9880, QCA9884, QCA9886, QCA9889, QCA9558, and QCA9563 chipsets, in addition to the support for Wi-Fi SON on IPQ401x and IPQ806x platforms that existed in previous releases.

## 19.1 Range Extender Placement and Auto-configuration

The Wi-Fi Self-Organizing Networks (Wi-Fi SON) feature set includes two new features targeted at simplifying the use of range extenders in the home. This page describes the design of these features, including how they can be customized (where applicable) by OEMs. It also provides a possible simple end user guide that could be provided with a range extender product to help users install such devices.

This feature is made up of a few components. These are:

- repacd init script
  - Responsible for Wi-Fi interface configuration and starting/stopping the necessary daemons.
- repacd daemon script
  - Responsible for monitoring the link conditions in order to update the LED states and trigger mode switches when appropriate.
- wsplcd daemon
  - Implementation of IEEE1905.1 Registrar and Enrollee functionality, enabling the cloning of Wi-Fi credentials to all interfaces.

The configuration options and the role of each of these daemons in the various features will be described further in the sections below.

### 19.1.1 Range Extender Automatic Configuration and Mode Switching

The range extender automatic configuration feature enables range extenders to be deployed within the home in a simple manner. This is one part of the whole solution for range extenders which also

includes the placement assistance as noted in [Section 19.1.2](#). The following general features are supported for automatic configuration:

- Automatic generation of an SSID and pass-phrase for an unconfigured device.
  - An unconfigured device is defined as one that either has no Wi-Fi interfaces defined or that has "OpenWRT" as the SSID without any encryption mode set on at least one interface.
- Configuring the Wi-Fi interfaces to support the appropriate backhaul (STA) and AP interfaces based on the operating mode
  - In QWRAP, ExtAP, and WDS modes, a single 5 GHz backhaul interface and both 2.4 GHz and 5 GHz AP interfaces.
  - In an exclusive Wi-Fi Self-Organizing Network (Wi-Fi SON) mode, backhaul and AP interfaces are created on all radios. This mode enables the multi-AP coordinated steering and Adaptive Path Selection features.
  - A configuration parameter controls the mode to be used. Note that some of the features are not fully supported in QWRAP and ExtAP modes currently.
- Automatic range extender mode switching:
  - Although a fixed mode of range extension can be selected, the best practice is to let the repacd daemon determine the mode to use based on the characteristics of the root AP. This auto mode is described in detail in the subsequent bulleted item.
- Automatic role switching:
  - For a device acting as a pure bridge (no router functionality), detection is performed to determine whether the device is connected to the gateway using the Ethernet interface or not, and to configure the Wi-Fi interfaces accordingly.
  - When the device is connected to the gateway using Ethernet interface (which can be a PLC dongle connected to its Ethernet port), the Wi-Fi STA interface is disabled and only the AP interfaces are enabled. Here, the assumption is the backhaul using Ethernet interface has more than sufficient capacity for any clients that may associate to the AP interfaces. This is generally the case with Gigabit Ethernet, although it may be less true with 100 Mbps Ethernet or PLC (depending on the standard and link conditions).
  - Otherwise, Wi-Fi is used for the backhaul interface.
- For a device whose primary purpose is a data consumption device (for example, a set top box), the device becomes a range extender if the conditions are suitable.
  - When the signal level from the root AP is strong enough but not so strong that coverage can be duplicated, the device transitions into range extender (RE) mode.
- While operating in WDS mode or full Wi-Fi SON mode, automatic cloning of the credentials for both 2.4 GHz and 5 GHz frequency bands on all enabled interfaces (both AP and STA interfaces) with a single button press.
  - This requires the root AP to be acting as an IEEE1905.1 Registrar. This functionality is provided by the wsplcd daemon.

These features are described in detail in the subsequent subsections.

### 19.1.1.1 Automatic Credential Generation

One of the roles of the repacd init script is to determine when it is being started with a pristine Wi-Fi configuration, and generate an SSID and passphrase for the initial Wi-Fi interfaces. As mentioned above, there are two possible configurations which are treated as a default configuration:

1. No Wi-Fi interfaces are defined (in other words, no wifi-iface sections in /etc/config/wireless).
2. One or more Wi-Fi interfaces with an SSID of OpenWRT and no security is enabled.

When one of these conditions is met, the SSID and passphrase are determined as follows (in the `_repacd_reset_default_config` function within the init script).

- **SSID**—The last three bytes of the bridge's MAC address are appended to the `whc-` string to form the SSID. For example, if the `br-lan` interface has a MAC address of `00:03:7F:1C:17:6B`, the corresponding SSID will be `whc-1C176B`.
- **Passphrase**—Eight bytes are read from `/dev/urandom` and then base-64 encoded to create a passphrase.

These typically are overwritten through the credential cloning process (so that the user's desired SSID and passphrase are present on the RE). However, this process ensures that each device contains a unique SSID and passphrase if the device is booted with this feature enabled.

At startup, if the repacd init script finds that the Wi-Fi interface configuration is not in its default state, the script looks for an existing SSID and passphrase to apply to all interfaces. The heuristics that the script uses are as follows:

1. Record the SSID and passphrase for the first interface seen while enumerating the interfaces.
2. If a STA interface is encountered while enumerating the interfaces, use the SSID and passphrase on the interface, instead of whatever was previously detected.

After the desired SSID and passphrase are determined, these attributes are applied to all of the interfaces. The following section describes how the interfaces are configured.

### 19.1.1.2 Configuration of Wi-Fi interfaces

The repacd init script is also responsible for configuring the Wi-Fi interfaces for range extension. It currently supports four modes of range extension—QWRAP, ExtAP, WDS, and full Wi-Fi SON. In the first three modes, a STA interface is created on the 5 GHz radio for the backhaul. AP interfaces are created on both 2.4 GHz and 5 GHz to provide extended coverage. In the last mode, a 2.4 GHz STA interface is also created. The mode that is used is controlled either by a configuration parameter, namely `repacd.repacd.ConfigGREMode` or by the `repacd` daemon when auto mode is used. Currently, auto mode is the default mode because the selection of the best mode is based on the type of central AP (CAP) that is detected.

The init script manipulates the wireless configuration parameters as appropriate to enable the desired operating mode. Any other unrelated wireless configuration parameters are not modified. This behavior enables OEMs to customize certain advanced settings and also allow the repacd init script to handle the basic configuration.

One value that can be managed by repacd is whether DFS channels are allowed or not. For the WDS and full Wi-Fi SON operating modes, DFS channels are allowed by default but may be disabled by setting the `repacd.repacd.BlockDFSChannels` config parameter to 1. Due to known limitations of the ExtAP and QWRAP modes at the time of development, DFS channels are always blocked regardless of this configuration parameter value.

### **QWRAP Limitations**

Configuring the same SSID and pass-phrase on the QWRAP AP interfaces as is used on the STA interface may encourage STAs to roam from the root AP to the range extender and vice versa. As QWRAP does MAC address translation, this generally leads to an IP address change if the STA attempts to renew its IP address. This can be disruptive to applications that have active traffic during such a roaming event.

For this reason, it is recommended that the QWRAP AP interfaces have a distinct SSID when compared to the STA interface. This is not currently handled in repacd but could be in the future.

Further more, QWRAP mode may also lead to a different MAC and IP address when a STA roams from one band to another on the same RE. To discourage STA roaming between bands during data activity, the recommendation is to use distinct SSIDs on each band.

#### **19.1.1.3 Automatic Range Extender Mode Switching**

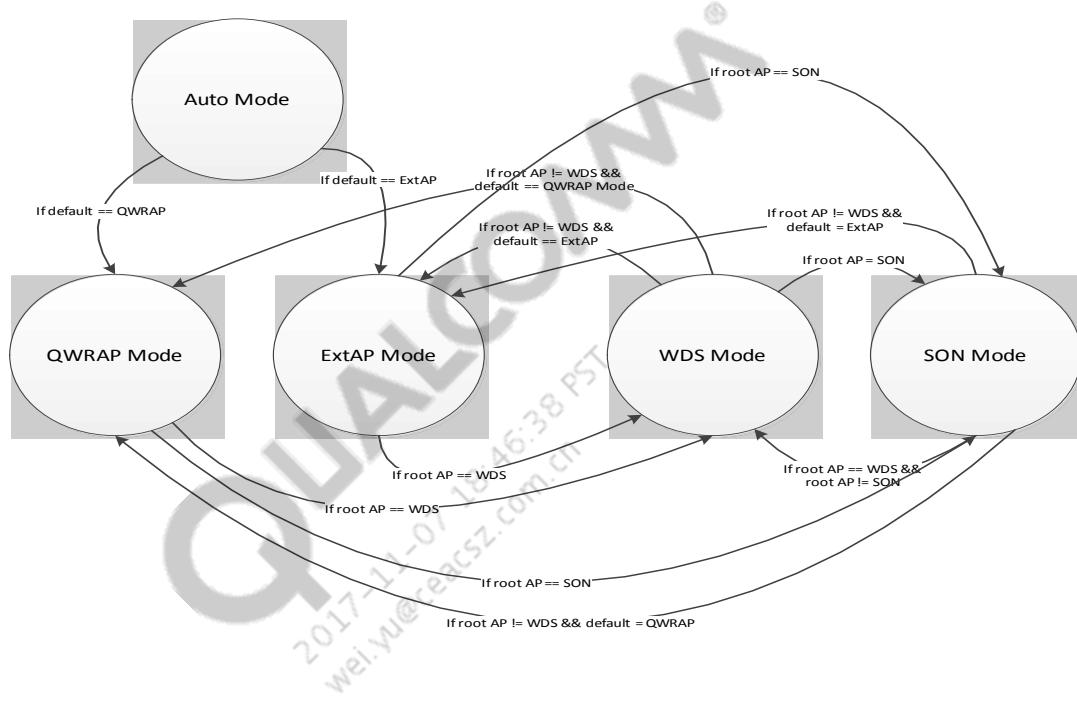
To support deploying range extenders in homes where there is an existing AP while still being able to provide the improved roaming and performance features of Wi-Fi SON if that AP is later upgraded by the consumer, **repacd** provides a feature where the method of range extension can be determined at runtime. In scenarios where WDS 4-address mode is likely not to work, it ensures either QWRAP or ExtAP mode (as selected by the OEM using the `repacd.repacd.DefaultREMode` config parameter) is used. If instead, the central AP (CAP) is operating in WDS or full Wi-Fi SON modes, **repacd** switches the RE into that mode and enable the more advanced features. The features that are supported in each mode are as follows:

- QWRAP or ExtAP
  - Interoperability with all commercial APs
- WDS
  - Single AP band steering on the RE itself. Because no coordination exists between the root AP and the range extender in this mode, the success rate of steering is limited to an extent. Generally, with 802.11v BSS Transition Management steering, the client typically ends up on the correct band, although it might end up switching APs. Client devices that are being steered using the legacy blacklist-based mechanism might end up on the other AP on the same channel, although the behavior is dependent on both the specific client device and the relative signal strengths of the BSSes that it sees.
- SON
  - Multi-AP coordinated steering
  - Adaptive Path Selection

This automatic mode switching is implemented in the **repacd** daemon. It operates by examining (through the use of new iwpriv commands) a vendor specific information element included in

association response by a Central AP that enables this automatic mode switching feature for range extenders. This IE allows the range extender to determine if the CAP is operating in WDS or full Wi-Fi SON mode. If it is, the RE itself will switch into the same mode by triggering a reconfiguration of the Wi-Fi interfaces and related services. If the CAP is not operating in these more advanced modes or does not include this IE at all, the range extender will use its desired inter-operable mode of range extension.

A state machine showing this mode determination is shown below.



These mode switching decisions are made at the time of association. If the CAP is being reconfigured without causing devices to disassociate, the range extender will not react to the new CAP mode.

#### 19.1.1.4 Guidelines for Wi-Fi SON on QCA9531 platforms

##### Configure APS for QCA9531 platforms running kernel version 4.4

The following UCI configurations are recommitted for adaptive path selection (APS) QCA9531 platforms that are running Linux kernel version 4.4 on both CAP and REs for improved throughput:

```

uci set hyd.PathChWlan.MaxMediumUtilizationForLC_W2='0'
uci set hyd.PathChWlan.MaxMediumUtilizationForLC_W5='99'
uci set hyd.PathChWlan.MaxMediumUtilization_W2='0'
uci set hyd.PathChWlan.MaxMediumUtilization_W5='99'
uci set hyd.hy.ConstrainTCPMedium='1'
uci commit
  
```

### Restart of hyd on QCA9531 platforms running kernel version 3.3.8

On QCA9531 platforms running Linux kernel version 3.3.8, a customized script is needed to monitor hyd, and to restart hyd automatically whenever it is killed. Because no support is available for procfs in kernel 3.3.8, script must be developed and owned by customers.

#### 19.1.1.5 Automatic Role Switching

The repacd feature provides some support for switching the role of the device in specific circumstances. The first circumstance is when the interface through which the device is connected to the gateway changes. Specifically, if the repacd daemon detects that the device is connected to the gateway via Ethernet, it will disable the STA interface. This detection is done in the repacd-gwmon.sh module. When an Ethernet link is detected, repacd pings the gateway, determines the MAC address of the gateway by examining the ARP table, and then checks the Ethernet switch's forwarding database (if necessary) to determine if that MAC address is present on one of its ports. If it is, it triggers the role change by invoking the repacd init script with the special `restart_in_cap_mode` target. On the other hand, if the gateway was previously detected on Ethernet and now can no longer be detected for three consecutive attempts or the Ethernet link goes down altogether, the repacd daemon will invoke the init script with the special `restart_in_noncap_mode` target. This will re-enable the STA interface to allow it to be used for the backhaul.

The other form of role switching is intended for devices like set top boxes that are primarily STA devices but could potentially act as range extenders as well. To use this form of role switching, set the repacd.repacd.DeviceType configuration parameter to Client. When this is done, when repacd is started, the init script will only enable the STA interface. Once an association is established, the RSSI is measured to determine if the device is placed in a suitable location for becoming a range extender. If the link is sufficient but is also not too strong, the repacd daemon will invoke the init script with the special `restart_in_re_mode` target.

This will enable the AP interfaces so that the device can then act as a range extender as well. It will remain acting as a range extender until it is power cycled, at which point a new assessment will be triggered.

#### Role Switching Limitations

If there are multiple devices that are connected to the gateway via Ethernet, the network will end up having multiple IEEE1905.1 Registrars. So long as the Wi-Fi credentials remain the same, this should not cause any significant issues, although it is not necessarily desirable. This will be addressed in a future release.

The Client device type is only currently intended to be used with the WDS and full Wi-Fi SON modes of range extension. It can conceivably be used in QWRAP mode if some other mechanism ensures the MAC address of the client device is properly configured in wrappd to enable upstream communication, although this has never been tested.

#### 19.1.1.6 Credential Cloning

One of the aspects of repacd that makes the life of the user simpler is the automatic cloning of credentials from the root AP to the range extender interfaces. When operating in WDS or full Wi-

Fi SON mode, the process fully configures both the STA and AP interfaces, resulting in all interfaces having an identical security configuration. This enables STAs to roam more seamlessly on their own from one BSS to another, as generally STAs will roam within an ESS much easier than they will between ESSes. This credential cloning is achieved through a multi-step process:

- The WPS push button method is used to establish the STA link to the root AP. This requires the user to push the WPS button on both the root AP and the range extender within a close period of time (2 minutes).
  - This step can be skipped if the range extender is being connected via Ethernet to the root AP.
- Once the STA link is established, the IEEE1905.1 AP Auto-configuration mechanism is used to clone the credentials onto the AP interfaces and add any additional settings to the STA interface.
  - This process relies on the root AP being configured as an IEEE1905.1 registrar with the necessary daemon (wsplcd) running there. This can be accomplished by enabling and starting repacd on the root AP.
  - The AP auto-configuration process essentially uses the WPS handshake messages (for example M1, M2, etc.) to obtain the credentials for the AP interfaces and then writes them back to UCI. In addition, by default deep cloning is enabled which copies a few other parameters, namely the channel of the root AP and the BSSID of the root AP. The latter ensures the device always connects to the root AP when there multiple APs from which to choose.
- As part of AP auto-configuration process, STA sends a vendor-specific IEEE1905.1 unicast message to the root AP confirming receipt of credentials and waits for another IEEE1905.1 vendor-specific multicast message to restart Wi-Fi services. To account for possible loss of the multicast message STA starts a local timer to restart Wi-Fi services.
- Root AP waits for the confirmation of credential cloning from all STAs for a short period of time. Every time it gets confirmation from a STA it extends the timer again to check if it receives any confirmation with remaining STAs. On expiry of this timer, root AP sends multiple IEEE1905.1 vendor-specific multicast message to indicate STAs to restart its Wi-Fi service. It also starts its own restart timer on expiry of which Wi-Fi service is restarted.

When QWRAP mode is being used, only the first step of setting the STA credentials can be performed, as running wsplcd with QWRAP is not currently supported. This means that the AP interfaces will still have to be manually configured after WPS completes or the repacd init script will have to be invoked with the restart target to clone the credentials onto the AP interfaces. See the important limitation regarding QWRAP about this above.

### Credential Cloning Limitations

In some cabled up testing, the credential cloning process can take an extended period of time. The root cause is unclear, but typically manifests as a large number of the IEEE1905.1 registrar search or response messages being lost. This can lead to a longer period of time during which the automatically generated credentials are used on the AP interfaces and potentially one additional Wi-Fi interface restart. However, the cloning will eventually complete at which point all interfaces will have the same credentials.

Due to the use of deep cloning which sets the desired BSSID on the STA interface to ensure the REs always connect to the CAP in a multi-RE network, if the central AP is changed, it will be necessary to perform another WPS push button operation to force the RE to associate to the new CAP. However, changes in the root AP's channel are handled seamlessly with no need to manually change any configuration.

For coordination of Wi-Fi service restart between root AP and STAs, it is essential to invoke wsplcd with restart\_after\_config\_change as parameter after wireless configuration has been changed on the root AP.

### 19.1.1.7 Configuration Parameters Summary

**Table 19-1** shows the configuration parameters that are relevant to the automatic configuration aspects of repacd.

**Table 19-1 repacd configuration parameters relevant for auto-configuration**

| Configuration Type | Section | Option            | Description   | Default |
|--------------------|---------|-------------------|---|---------|
| Config             | repacd  | Enable            | Whether the RE placement and auto-configuration logic is enabled or not   | 0       |
| Config             | repacd  | ManagedNetwork    | The name of the network where the Wi-Fi interfaces being managed will reside.<br>Note that currently repacd is limited to only a single network.  | lan     |
| Config             | repacd  | DeviceType        | The primary role of the device. Must be one of RE or Client.  | RE      |
| Config             | repacd  | Role              | The current role (CAP or NonCAP) for this device.<br>This should generally not be changed directly, as the value is set by the init script and read by the daemon.  | NonCAP  |
| Config             | repacd  | ConfigREMode      | The mechanism to use for range extension. Supported values are: wds and qwrap<br><br><b>NOTE</b> QWrap mode does not support the full credential cloning logic.   | wds     |
| Config             | repacd  | BlockDFSCchannels | Whether to disable DFS channels when creating AP interfaces.  | 0       |
| Config             | repacd  | LinkCheckDelay    | The amount of time (in seconds) to wait between successive link checks.<br><br><b>NOTE</b> The actual amount of time between two link checks may be 1 second larger than this (due to implementation considerations). | 2       |

There are also a few configuration parameters for wsplcd that are relevant to the auto-configuration process, as listed in [Table 19-2](#).

**Table 19-2 wsplcd configuration parameters relevant for auto-configuration**

| Configuration Type | Section | Option                    | Description  | Default |
|--------------------|---------|---------------------------|--|---------|
| Wsplcd             | config  | SSIDSuffix                | Suffix to append to the SSID when cloning onto the AP interfaces.<br>For the best roaming performance in WDS mode, this should be left as the empty string so that STAs can roam freely between the root AP and RE.  |         |
| Wsplcd             | config  | DeepClone                 | Whether the channel and BSSID should be copied in addition to the security parameters (SSID, passphrase, encryption mode) when cloning credentials.  | 1       |
| Wsplcd             | config  | WaitOtherBandSecs         | How many seconds to wait for the cloning to complete on the second band after the first one completes before applying the changes.<br>Generally the second band should complete soon after the first band.   | 20      |
| Wsplcd             | config  | WaitFirstBandSecs         | How many seconds to wait for the credential cloning to complete on at least one band.<br>During this period, the daemon will send a search message every 2 seconds. After this time elapses, it will switch to less frequent searches (every 60 seconds).      | 30      |
| Wsplcd             | config  | DebugLevel                | Verbosity of debug logs.<br>For example. DEBUG, INFO, ERROR.   |         |
| Wsplcd             | config  | WriteDebugLogFile         | Whether to clear the contents of the log file before writing, or to append to the existing file.<br>For example, APPEND, TRUNCATE. If using APPEND mode, user should note that the size of log file will keep growing and can fill up all RAM.                 |         |
| Wsplcd             | config  | ConfigRestartLongTimeout  | This Timeout value (in seconds) is used by range extender and the timer is started after sending confirmation of the reception of credential and while waiting for message from root AP to indicate Wi-Fi service restart.                                     | 20      |
| Wsplcd             | config  | ConfigRestartShortTimeout | This timeout value (in seconds) is used by root AP and the timer is started/restarted upon receiving confirmation message from range extender. On expiry of this timer, root AP sends out multiple multicast message indicating STAs to restart Wi-Fi service. | 5       |
| Wsplcd             | config  | ConfigApplyTimeout        | This timeout value (in seconds) is used by range extender to delay the restart of Wi-Fi service so as to ensure all multicast messages are sent out on bridge before restarting Wi-Fi service.   | 10      |

### 19.1.1.8 Wsplcd daemon in Coordinated ATF between RAP and Repeater

On starting the ‘wsplcd’ daemon, it reads ATF configurations and sends it across to the respective Repeater as a 1905 Data packet. The daemon running on the Repeater end would receive this

packet and would configure ATF. Note that for any change in ATF configurations to be applied (send across to Repeater), the wsplcd daemon at the RootAP need to be restarted.

The ATF configurations can be viewed at the repeater end through the existing command ‘wlanconfig <vapname> showatftable’.

To support this feature, new 1905 Vendor types are added. It is expected that the user is aware of the Repeater MAC address, available list of SSIDs and the wifi device or VAP name of the Repeater that he would intend to configure. Intelligent mapping of SSIDs/VAPs is not supported. The configurations entered at the RootAP will be send across to Repeater AP. All existing error checks for ATF configurations will be considered at the Repeater AP end before applying the configuration. The Repeater AP should be connected to the RootAp before sending the configuration or else the configurations would be received by the Repeater AP.

The Coordinated ATF feature is designed keeping in mind that it can be further extended in future. The following features are not supported:

- Support to read ATF configuration of the Repeater AP
- One shot configuration request to configure all Repeater AP’s with the same configuration
- Intelligent mapping of devices/SSID are not supported in the first version.

## 19.1.2 Range Extender Placement Assistance

### 19.1.2.1 Design

The range extender placement assistance consists of LED indications to direct the user to a suitable location for the RE. The basic idea is to determine at boot time whether the RE is in a location where it has both a sufficient link to the CAP (so as to not artificially limit the throughput on devices that happen to select the RE instead of the CAP) and where it is not just duplicating the CAP’s existing coverage and thus not providing much benefit to the end user in resolving a coverage hole. This is done by the repacd-wifimon.sh module of the repacd daemon. At a high level, once the association completes, it starts a ping to the GW to ensure the RSSI values get updated. It then periodically samples the RSSI and computes an average value (although this will be converted to a median in the future). One of three LED patterns is then used to indicate to the user to move the RE closer to the CAP, farther from the CAP, or leave it where it is as the location is suitable for range extension.

The state machine that drives this LED configuration process is shown in [Figure 19-1](#) and [Figure 19-2](#).

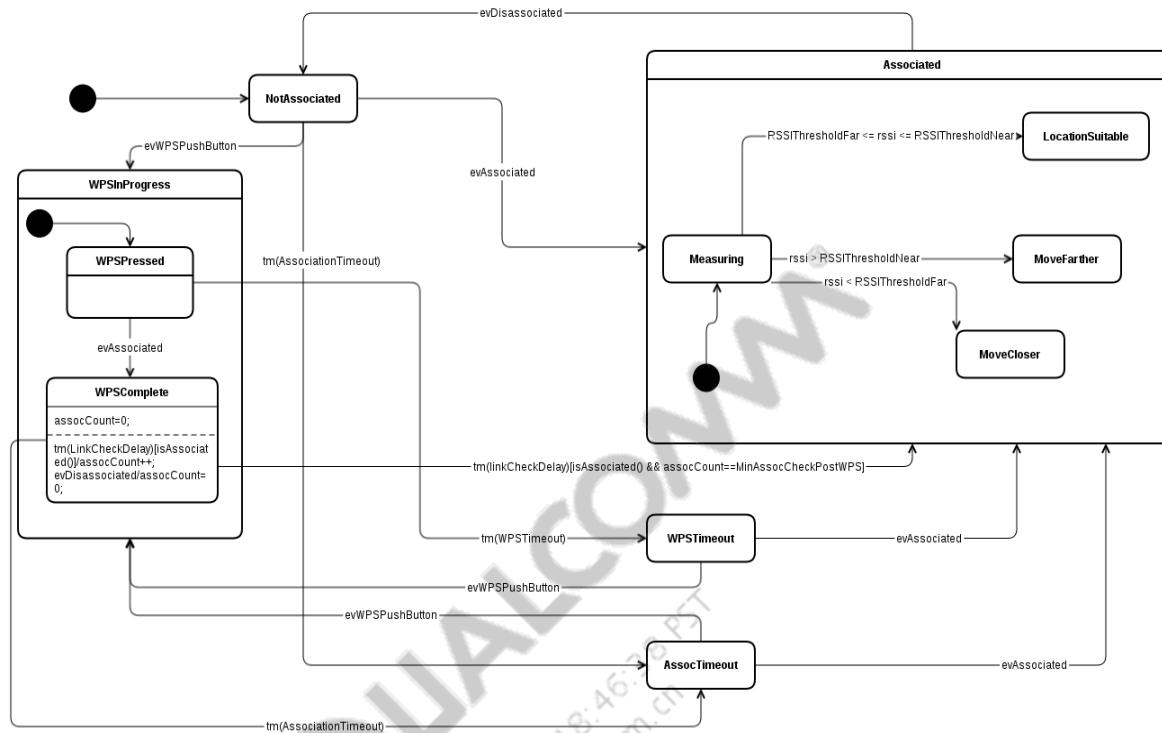
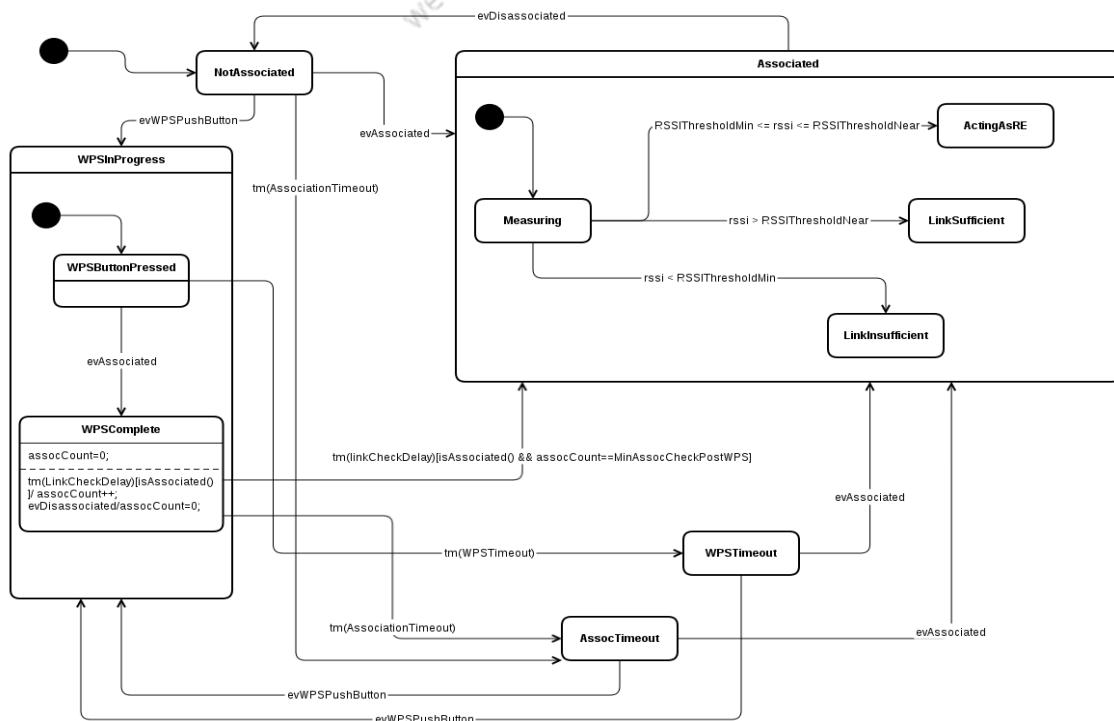


Figure 19-1 State machine for LED configuration process on RE devices



### Figure 19-2 State Machine for LED Configuration Process on Client Devices

Some of these states have some special logic which necessitate a further explanation. Firstly, the WPSInProgress state is composed of two sub states. When the button is pressed, the state machine first waits for an association to complete on the backhaul interface. Once this occurs, it moves into a new sub-state where it ensures the link is stable before beginning the measurement process. It does this by confirming for MinAssocCheckPostWPS times (separated by LinkCheckDelay) that the STA interface indicates it is associated. If the association is lost (which typically will occur during the credential cloning), the counter is reset back to 0. If the association is not stable after AssociationTimeout seconds after the association, the RE is declared to be too far away.

The other special state is the Measuring state. In this state, a background ping to the gateway IP address is started and then the downlink RSSI is sampled every LinkCheckDelay seconds. The measurements are then averaged over RSSINumMeasurements and this value is used for comparison to the thresholds, which ultimately lead to the final LED states.

### LED Behavior with Role and Mode Switches

When using the automatic range extender mode switching or with the client device type, the state machine may traverse through the transitions two or three times. This is due to the daemon getting restarted during role switches and the association getting lost during the Wi-Fi restart after credential cloning. Thus, from a factory default state, the transition sequence for a client device that ends up acting as an RE could be NotAssociated ->WPSInProgress -> Measuring -> ActingAsRE -> NotAssociated -> Measuring -> ActingAsRE -> NotAssociated -> Measuring -> ActingAsRE.

#### 19.1.2.2 LED Schemes

Two LED schemes have been defined, one which uses two distinct single color LEDs and one that uses a single 2-color LED. These are both examples of what is possible and can potentially be tweaked by OEMs depending on their platform requirements. These schemes are shown in [Table 19-3](#) and [Table 19-4](#) (for AP148 and similar platforms):

**Table 19-3 Example LED schemes for RE devices**

| RE State            | 2-LED scheme (default)                         | 1-LED schme                                  |
|---------------------|--|--|
| NotAssociated       | Blink USB-I at 1 Hz                            | Blink green at 1 Hz                          |
| WPSInProgress       | Blink USB-I at 2 Hz                            | Blink green at 2 Hz                          |
| Measuring           | Blink USB-I and USB-II at 2 Hz                 | Blink green at 5 Hz                          |
| WPSTimeout          | Blink USB-I with 2 seconds on and 1 second off | Blink red with 2 seconds on and 1 second off |
| AssocTimeout        | Blink USB-I with 5 seconds on and 1 second off | Blink red with 5 seconds on and 1 second off |
| RE_MoveCloser       | USB-I solid                                    | Solid red                                    |
| RE_MoveFarther      | USB-II solid                                   | Blink red at 1 Hz                            |
| RE_LocationSuitable | USB-I and USB-II solid                         | Solid green                                  |

**Table 19-4 Example LED schemes for client devices**

| RE State          | 2-LED scheme (default)                         | 1-LED schme                                  |
|-------------------|--|--|
| NotAssociated     | Blink USB-I at 1 Hz                            | Blink green at 1 Hz                          |
| WPSInProgress     | Blink USB-I at 2 Hz                            | Blink green at 2 Hz                          |
| Measuring         | Blink USB-I and USB-II at 2 Hz                 | Blink green at 5 Hz                          |
| WPSTimeout        | Blink USB-I with 2 seconds on and 1 second off | Blink red with 2 seconds on and 1 second off |
| AssocTimeout      | Blink USB-I with 5 seconds on and 1 second off | Blink red with 5 seconds on and 1 second off |
| CL_LinkSufficient | USB-I solid                                    | Solid green                                  |
| CL_LinkInadequate | USB-II solid                                   | Blink red at 1 Hz                            |
| CL_ActingAsRE     | USB-I and USB-II solid                         | Solid green                                  |

### Single LED Scheme Limitations

Note that with the single LED scheme, it is not possible to distinguish between the link sufficient and acting as RE cases for the client mode of operation. This was chosen to avoid having the LED blink when in one of the desired states but could be changed by an OEM if so desired.

The two LED scheme is the default one that comes out of the box. Switching between the schemes requires updating the /etc/config/repacd file. The file to use for the single LED scheme can be found in the source tree under qSDK/qca/feeds/whc/qca-whc-repacd/files/repacd.config.status-led. The LEDs will also need to be defined in the UCI system configuration using the following set of commands at a shell:

```
uci set system.led_status_green=led
uci set system.led_status_green.name='Status_Green'
uci set system.led_status_green.sysfs='ap148:green:status'
uci set system.led_status_green.trigger='none'
uci set system.led_status_red=led
uci set system.led_status_red.name='Status_Red'
uci set System.led_status_red.sysfs='ap148:red:status'
uci set system.led_status_red.trigger='none'
uci commit system
```

### 19.1.2.3 Configuration Parameters

Configuration parameters are provided to control when the transitions between states should occur and how the LEDs are updated for each state. **Table 19-5** describes these parameters, many of which have been mentioned above in the state machines.

**Table 19-5 Parameters for LED indications**

| Configuration Type | Section  | Option                       | Description   | Default |
|--------------------|----------|------------------------------|---|---------|
| WiFiLink           | WiFiLink | MinAssocCheckPostWPS         | The number of times the association must be deemed up after a WPS button press before it is considered stable enough before an RSSI measurement can begin.  | 5       |
| WiFiLink           | WiFiLink | MinAssocCheckPostBSSIDConfig | The number of times the association must be deemed up after a BSSID is configured before it is considered stable enough before a rate measurement can begin.  | 5       |
| WiFiLink           | WiFiLink | WPSTimeout                   | The amount of time (in seconds) to wait for an association to take place after the WPS button is pressed.<br><br>If this amount of time elapses without the STA interface associating, the device will be assumed to be too far from the CAP. | 180     |
| WiFiLink           | WiFiLink | AssociationTimeout           | The amount of time (in seconds) to wait for the STA interface to associate before considering the device as too far from the CAP.<br><br>Note that a WPS push button cancels this timer and runs the WPS timeout instead.                     | 300     |
| WiFiLink           | WiFiLink | RSSINumMeasurements          | The number of measurements to take to arrive at an average RSSI to compare against the near/far thresholds.   | 5       |
| WiFiLink           | WiFiLink | RSSIThresholdFar             | The signal level (in dBm) below which the RE is considered too far from the CAP and should be moved closer.   | -75     |
| WiFiLink           | WiFiLink | RSSIThresholdNear            | The signal level (in dBm) above which the RE is considered too close to the CAP and should be moved farther.  | -60     |
| WiFiLink           | WiFiLink | RSSIThresholdMin             | The signal level (in dBm) above which a device whose primary role is as a client is eligible to become a range extender (so long as it does not exceed RSSIThresholdNear).  | -75     |
| WiFiLink           | WiFiLink | DaisyChain                   | The Daisy chain feature which allows a RE to connect to another RE or the Root AP, this enables Best uplink selection also.   | 0       |
| WiFiLink           | WiFiLink | RateNumMeasurements          | The number of measurements to take to arrive at an average Rate to compare against the Min/Max thresholds.  | 5       |
| WiFiLink           | WiFiLink | RateThresholdMin5GInPercent  | The 5 GHz rate level (in Mbps) below which the RE is considered associated with bad link and it will trigger the best uplink selection logic to find suitable AP.   | 40      |

**Table 19-5 Parameters for LED indications**

| Configuration Type | Section  | Option                                 | Description  | Default |
|--------------------|----------|--|--|---------|
| WiFiLink           | WiFiLink | RateThresholdMax5GInPercent            | The 5 GHz rate level (in Mbps) above which the RE is considered associated with bad link and it will trigger the best uplink selection logic to find suitable AP.  | 70      |
| WiFiLink           | WiFiLink | RateThresholdPrefer2GBackhaulInPercent | The 5 GHz rate level (in Mbps) below which the RE is considered associated with bad link and it will bring down 5G backhaul and brings up 2.4G backhaul.   | 5       |
| WiFiLink           | WiFiLink | 5GBackhaulBadlinkTimeout               | The timeout (in seconds) for which if the RE is associated with bad link then it will bring down 5G backhaul and brings up 2.4 GHZ backhaul.   | 60      |
| WiFiLink           | WiFiLink | BSSIDAssociationTimeout                | The timeout (in sec) for which if the RE is not associated with configured BSSID then it will clear the configured BSSID and then restarts the RE for fresh association.<br><br>Random connectivity issues might be observed with daisy-chain RE after reboot. These problems occur with the default BSSID value of 90. If a DFS channel is used as backhaul, it is recommended to use 120 seconds as the timeout value using uci set repacd.WiFiLink.BSSIDAssociationTimeout=120 command. | 90      |
| WiFiLink           | WiFiLink | RateScalingFactor                      | The scaling factor (in percent) to be used in this RE for best uplink selection algorithm.   | 70      |
| WiFiLink           | WiFiLink | 2GBackhaulEvalTime                     | Evaluation time (in sec) for bring back up forcibly down 2.4 GHz backhaul  | 1800    |
| LEDState           | Varies   | Name_1<br>Name_2                       | The name of the LED configuration section (in /etc/config/system) to use to resolve this to a SysFS name.  | -       |
| LEDState           | Varies   | Trigger_1<br>Trigger_2                 | The mode in which the LED should operate.<br>none - Solid on or off<br>timer - Blinking  | -       |
| LEDState           | Varies   | Brightness_1                           | The value to set for the LED brightness.<br><br>At least on AP148, the brightness does not seem to matter, so a value of 1 should be used for on and a value of 0 for off.   | -       |
| LEDState           | Varies   | DelayOn_1<br>DelayOn_2                 | The amount of time (in milliseconds) the LED should stay on.<br><br>This is only relevant if the corresponding trigger is set to timer.  | -       |
| LEDState           | Varies   | DelayOff_1                             | The amount of time (in milliseconds) the LED should stay off.<br><br>This is only relevant if the corresponding trigger is set to timer.   | -       |

The LEDState sections are the means to configure different LED schemes. The section name can take one of the following values:

- NotAssociated - STA interface is still trying to associate
- WPSInProgress - WPS button was pressed and the timeout has not yet occurred

- Measuring - STA is associated and an average downlink RSSI value is being computed
- WPSTimeout - Failed to establish an association within WPSTimeout seconds
- AssociationTimeout - Failed to establish an association within AssociationTimeout seconds
- RE\_MoveCloser - RSSI is too weak or the STA was unable to associate
- RE\_MoveFarther - RSSI is too strong (duplicating coverage)
- RE\_LocationSuitable - RSSI is sufficient for the backhaul without too much coverage overlap
- CL\_LinkSufficient - RSSI is sufficient for the device to act as a client device but is not sufficient for it to become a range extender.
- CL\_LinkInadequate - RSSI is too weak or the device cannot even associate.
- CL\_ActingAsRE - RSSI is in the sweet spot to allow the device to act as an RE while continuing to meet the client requirements.

This allows the LED scheme to be tweaked through the configuration file. The value names have a suffix to allow for up to 2 LEDs to be controlled in a given state. All parameters with the same suffix apply to the same LED. All values, except for the Name\_1 and Name\_2 are directly written to SysFS files as described in the Linux kernel documentation.

### 19.1.3 Using repacd

As mentioned above, most of the repacd logic is applicable primarily to devices that are not acting as a gateway. This is because gateway devices are assumed to be acting as a root AP (CAP) and thus do not need any of the auto-configuration or placement logic. As the default QSDK image provides a gateway configuration, it is first necessary to reconfigure devices that are intended as range extenders to act as a pure bridge. This would typically be done in the factory for a range extender device, although it could also be done via a web interface option. The following series of commands should be used:

```
# Configure DHCP to not respond
uci set dhcp.lan.ignore=1; uci commit dhcp;
/etc/init.d/dnsmasq restart
# Disable NAT - Dynamic IP address
uci set network.lan.ifname='eth0 eth1'; uci set network.lan.proto=dhcp;
uci delete network.wan; uci commit network
/etc/init.d/network restart
# Disable NAT - Static IP address (update the IP addresses as
appropriate)
uci set network.lan.ifname='eth0 eth1';
uci set network.lan.proto=static; uci set network.lan.ipaddr=192.168.1.3;
uci set network.lan.gateway=192.168.1.1; uci set
network.lan.dns=192.168.1.1;
uci delete network.wan; uci commit network
#disable mcsd.
uci set mcsd.config.Enable=0
/etc/init.d/network restart
/etc/init.d/dnsmasq restart
```

Once the CAP and RE are configured as desired from a networking perspective, all that is left to do is to enable and start repacd. For WDS operation, perform the following:

```
uci set
repacd.repacd.Enable=1
uci commit repacd
/etc/init.d/repacd start
```

On the CAP, this should be done after setting the SSID and passphrase as desired and any other ‘advanced’ Wi-Fi settings that are desired. On the RE, no Wi-Fi settings need to be done, as that will be handled during the push button process.

When testing with a fixed range extender mode (instead of auto mode), an additional step is needed to set the mode prior to starting repacd. Below is an example of forcing the RE to operate in QWRAP mode:

```
uci set repacd.repacd.REMode=qwrap
uci commit repacd
```

When testing with ethernet monitoring mode, an additional step is needed to enable it:

```
uci set repacd.repacd.EnableEthernetMonitoring=1
uci commit repacd
```

#### 19.1.4 Viewing Logs

The various state transitions of repacd and its assessment of the Ethernet and Wi-Fi link state/quality are logged to the system logger. These messages can be viewed with the logread -f | grep repacd command.

#### 19.1.5 Sample scenario on LEDs placement

Below are some sample instructions that could be included with a range extender where the automatic configuration and placement logic is enabled. These could be accompanied by diagrams that show the user which LEDs are being used to indicate placement. Note that this procedure is written as if the LEDs were labeled according to their purpose, although on existing platforms (for example, AP148), they are labeled USB-I (for MoveCloser) and USB-II (for MoveFarther).

- Find the button on your main access point that is labeled WPS. This will be used in a subsequent step.
  - Some access points may not have a physical button and may instead require the user to log in via its web interface and activate WPS there. See the manual for your main access point for more details.
- Plug the range extender into an outlet in the same room as your main access point and wait for the MoveCloser LED to start blinking.
- Press the WPS button on the range extender and the main access point within one minute of each other.
- Wait for the MoveCloser and MoveFurther LEDs to stop blinking. Generally the MoveFurther LED will be solid and the MoveCloser LED will be off if the devices are in the same room.
- If the MoveCloser LED starts to blink in a pattern where it is on for 2 seconds and off for 1 second, this indicates the WPS process failed. Return to step 3 and try again.

- Unplug the range extender and move it to a new location, approximately half way between the primary access point and a location in your home where you have poor Wi-Fi coverage.
- Plug the device in again.
- Wait for one or both LEDs to become solid.
  - If only the MoveFarther LED is lit, this location is still too close to the main access point. Try moving it to an adjacent room that is farther from the main access point and return to step 6.
  - If only the MoveCloser LED is lit, this location is too far from the main access point and will lead to poor performance. Try moving it to an adjacent room that is closer to the main access point and return to step 6.
  - If the MoveCloser LED is lit for 5 seconds and then is off for 1 second, this means the range extender is unable to connect to the main access point. Try moving it to an adjacent room that is closer to the main access point and return to step 6.
  - If both LEDs are lit, this location is appropriate for the range extender. Leave the device there and enjoy your newly improved Wi-Fi coverage.

**NOTE** With Wi-Fi SON enabled, peak throughput drops by 10-15%. This drop in throughput is an expected behavior.

**NOTE** With Wi-Fi SON deployments (with five REs) in high-interference environments, a condition is observed where client laptops might lose Internet connectivity after 25-30 hours of continuous streaming. A reset or a reboot of the CAP reverts this condition.

## 19.2 Wi-Fi SON: Daisy chain and Best Uplink selection



Daisy Chain Support can be used to extend the Wi-Fi Range beyond the RootAP and the repeater. Additional repeaters or range extenders (RE) can be added to existing repeaters as shown in the preceding figure. As the network grows beyond a simple star topology, an effective scan algorithm is needed at the RE to find the best uplink AP.

This section describes the daisy chain topology for Wi-Fi SON. The daisy chain network illustrated in this section consists of a root AP and two REs (called daisy chain lite), although the daisy chain deployment can be extended to multiple REs (called daisy chain full support)

## 19.2.1 Information Elements

### 19.2.1.1 Root AP

The Wi-Fi SON advertisement IE is added as part of Beacon, Probe response and Association response frames. Addition of this SON IE to the above frames is applicable only in SON/WDS mode.

Root AP advertises SON capabilities using Wi-Fi-SON IE – extension of vendor IE.

1. Wi-Fi-SON Version Number (0x1)
2. Capabilities (SON and WDS 0x3)
3. Hop Count (0 for Root AP)
4. Root AP indicator (0x1)

### 19.2.1.2 REs

The Wi-Fi SON advertisement IE is added as part of Beacon, Probe response and Association response frames of AP VAP. Addition of this SON IE to the frames is applicable only in SON/WDS mode. Connected REs only broadcast this IE as part of the frames. Also, this SON IE will be added as part of Association request of STA VAP.

Range Extenders (Repeater APs) advertises SON capabilities using Wi-Fi SON IE – extension of vendor IE.

1. Wi-Fi-SON Version Number (0x1)
2. Capabilities (WDS 0x1)
3. Hop Count (distance from Root AP)
4. Root AP indicator (0x0)
5. Number of connected REs

### 19.2.1.3 Daisy chain

Daisy chaining enables a range extender (RE) to connect to another RE or the root access point (AP), unlike the case in star topology. Daisy chaining uses the best BSSID selection feature for management of links.

#### Operating Modes

Central AP (CAP) and all REs automatically derive the mode, based on their configuration or the serving AP mode.

#### Hop 1 - SON Mode

All REs at a distance of one hop from CAP (Root AP) use the existing SON mode of repacd. These REs must enable managed backhaul and managed AP on both 2.4 GHz and 5 GHz radios (SON

mode). All the SON features such as AP steering and band steering are active, similar to a star topology. The bridge looping is prevented by the hyd daemon.

### **Hop 2 and beyond - DAISY\_SON Mode**

All REs that are a distance of two or more hops enable managed backhaul only on 5 GHz radio and managed AP on both 2.4 GHz and 5 GHz radios. A new submode is created for SON called DAISY\_SON. If the managed 5 GHz backhaul link is associated, and if the rate is lower the configured preferred rate or not associated for the configured timeout, a managed 2 GHz backhaul link is created and the 5G backhaul link is disabled.

While starting operations in the managed 2 GHz backhaul link, rate measurements are disabled and the best uplink algorithm is not enabled. Because only one managed backhaul link is present, bridge looping is avoided. AP steering is enabled in the hyd daemon in this mode.

### **Push Configuration**

The credential configurations are pushed by WSPLCD daemon. BSSID configurations that are pushed by WSPLCD as part of credential configuration are disabled. For REs at a distance of one hop, because of the presence of two managed backhaul links, the RE identifies the partner 2 GHz BSSID after using the best uplink selection algorithm on the managed 5G backhaul link. This method of identification ensures that both (2.4 GHz and 5 GHz) managed backhaul links of RE connect to the CAP only.

For REs at a distance of two or more hops, because of the presence of only one managed backhaul link, the backhaul BSSID is determined and pushed to the configuration file using the best uplink selection algorithm.

#### **19.2.1.4 Best uplink selection**

The root AP and all REs that are in SON or DAISY\_SON mode will advertise the SON IE in its Beacon, Probe response, and Association response frames. All REs depend on this IE information to determine the best uplink AP. But Root AP is an independent entity which comes up on its own. Best uplink selection runs only on managed 5 GHz backhaul.

Whenever a RE comes up, it shall scan for available Root AP and REs and it will decide on best BSSID to connect, based on default BSSID ranking. Once the RE is associated, it will start rate measurement, Best uplink selection, after association and operating mode is found stable. It also advertises the IE in its Beacon, Probe response and Association response frames of all AP VAPs.

The RE will send the SON IE in Association request frame so that Receiving AP knows whether the connected STA is a RE or normal client. This data can be later used in our algorithm, if required.

REPACD daemon will periodically scan for current BSSID Rate level, and if it is not within configured 5G\_RATE\_MIN and 5G\_RATE\_MAX threshold then REPACD will trigger the best uplink selection algorithm on managed 5G backhaul link. If selected BSSID is different from current BSSID, selected BSSID will be written to configuration file and Wi-Fi restart will be triggered so that it associates to preferred link.

**NOTE** Operating mode switch can happen due to BSSID selection.

## Best uplink algorithm

The best BSSID selection algorithm is implemented in driver for STA VAP. Before the introduction of this algorithm, REPACD triggered a scan. This algorithm uses the driver scan entries as input and determines the best BSSID. The selected BSSID is used by the REPACD daemon to configure the managed 5G backhaul to associate to the BSSID from currently associated BSSID. This best uplink algorithm through IOCTL interface.

Any RE's rate is estimated using its received beacon RSSI, 5G backhaul rate (IE broadcasted in beacon), distance from Root AP (hop count) and user configured rate scaling factor.

The rates will be arrived from rate estimator table, same table used in LBD.

$\text{uplink\_rate} = (\text{rate\_current} * \text{rate\_uplink} * (\text{scaling\_factor})^{\text{root\_distance}}) / (\text{rate\_current} + \text{rate\_uplink})$

where,

$\text{rate\_current}$  = estimated from scan entry of received beacon RSSI (lookup table from LBD)

$\text{rate\_uplink}$  = advertised in the SON IE of received beacon

$\text{scaling\_factor}$  = user configurable

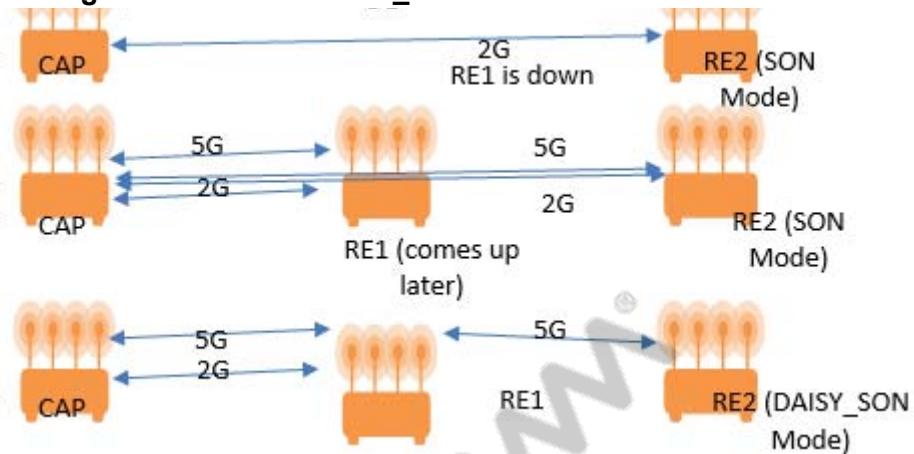
$\text{root\_distance}$  = advertised in the SON IE of received beacon

Estimated "*uplink\_rate*" is compared between all the interested scan entries and best BSSID is arrived.

By default, 2.4 GHz STA VAP is disabled. After the 5 GHz link is associated, the 2.4 GHz BSSID is extracted from the beacon of the AP to which the 5GHz STA interface is associated and configured (in wireless config file). After this configuration occurs, this 2.4 GHz STA interface is enabled and connects with the uplink node. This methodology ensures that 5G and 2.4G STAs connect to the same uplink node.

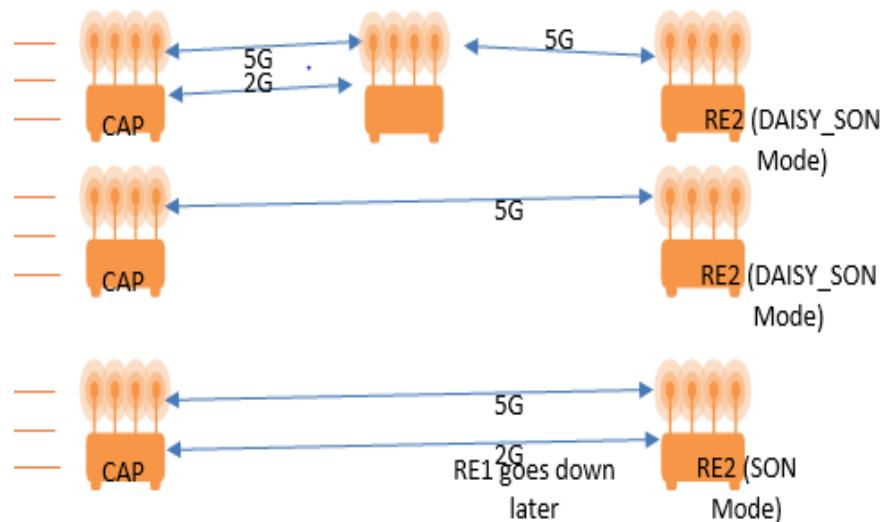
### 19.2.1.5 Daisy Chain Scenarios

This section describes sample configuration scenarios in which the daisy chain operation and best uplink algorithm can be implemented.

**RE mode change from SON to DAISY\_SON mode****Figure 19-3 RE Movement from SON to DAISY\_SON Mode**

Consider the scenario in which an RE, RE2, is connected to a central AP (CAP), when another RE, RE1, is down. Later, when RE1 comes up again, the following processes occur to transition RE2 from SON mode to DAISY\_SON mode.

1. RE1 connects to CAP in SON mode.
2. Best uplink algorithm on the 5 Ghz of RE detects that RE1 is better.
3. BSSID of RE1 is configured in 5 GHz backhaul of RE2, and the network is restarted.
4. After RE2 connects to RE1, it discovers that RE1 is at hop 1.
5. RE2 switches to DAISY\_SON mode (disables 2 GHz) and the network is restarted again.

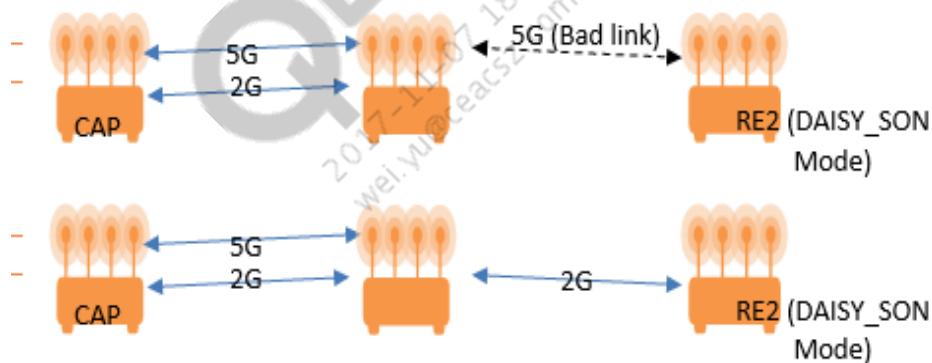
**RE Movement from DAISY SON to SON Mode****Figure 19-4 RE Movement from DAISY SON to SON Mode**

Consider the scenario in which RE1 is connected to CAP and RE2 is connected to RE1. Assume that RE1 goes down after a certain period of time. The following processes occur to transition RE2 from DAISY SON mode to SON mode.

1. RE2 loses connection with RE1.
2. Because the BSSID of RE1 remains configured, RE2 repeatedly looks up for RE1 until a timeout (1 minute) occurs.
3. After the timeout, BSSID of RE1 is removed from the configuration of RE2.
4. The network of RE is restarted to discover any AP.
5. RE2 discovers the CAP and associates with the CAP in DAISY SON Mode.
6. RE2 discovers that it is at hop 1; so it restarts the network in SON mode (enables 2 GHz).
7. RE2 also scans for CAP in the 2 GHz band.
8. If BSSID of CAP is different from the 2 GHz AP with which the AP is currently associated, the BSSID of CAP is configured for 2 GHz and network is restarted.

### **Bad link in hop2 of the RE 5 GHz**

#### Hop2 RE 5G has a Bad link

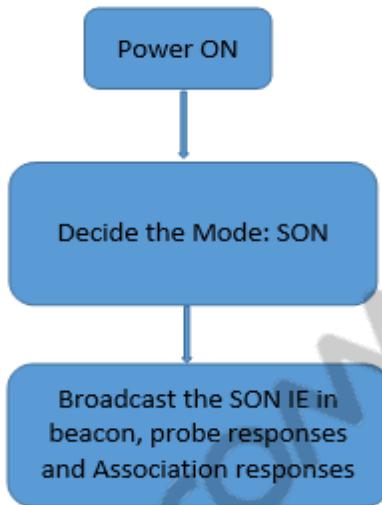


**Figure 19-5 Hop2 RE 5 GHz has a Bad link**

Consider the scenario in which the 5 GHz link of RE2 is bad. It is preferred that the 2 GHz link (which is currently disabled) can be used instead of 5G. The following process occur at RE2 to use the 2 GHz radio.

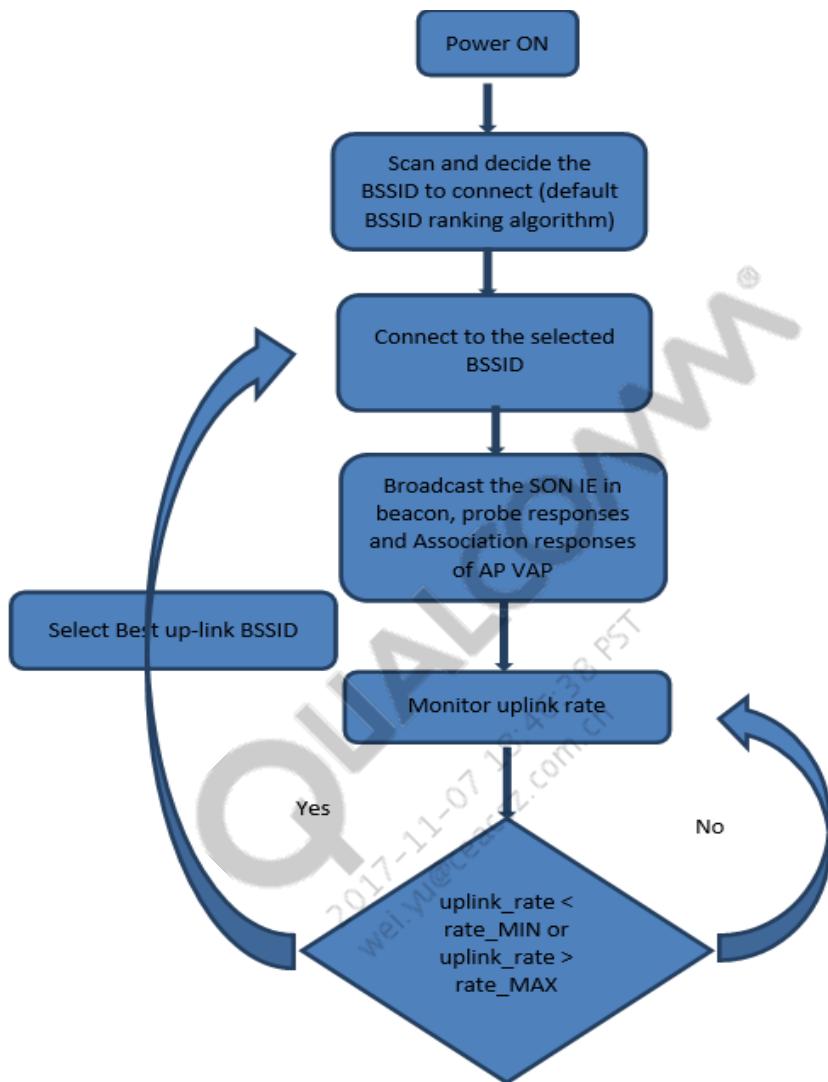
1. RE2 starts a timer for the bad 5 GHz link.
2. If a better AP is discovered by the best uplink algorithm, RE2 connects to a different RE.
3. If the 5 GHz bad-link timer expires, then 5G is brought down forcibly and 2G is enabled.

### 19.2.1.6 Root AP Flow



QUALCOMM®  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

### 19.2.1.7 RE Flow diagram

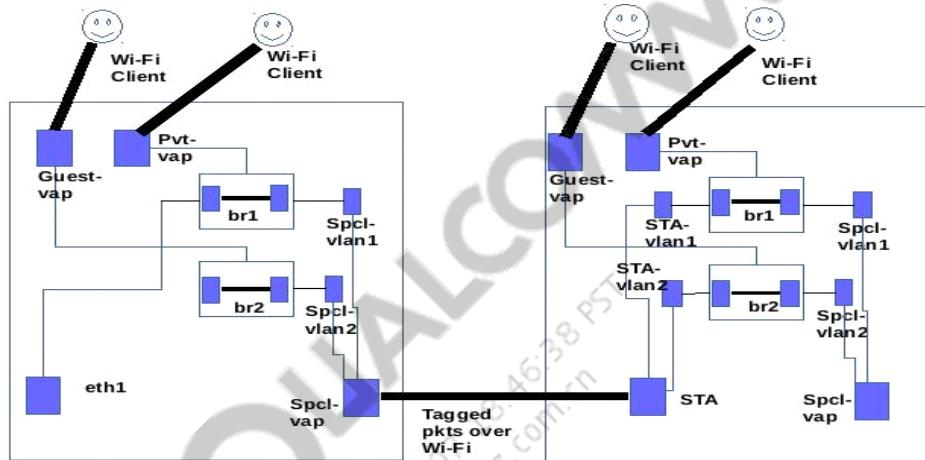


## 19.3 Wi-Fi SON: Multi-SSID and Traffic Separation

This feature is used to provide multi-SSID and traffic separation support on Root and Repeater AP when operating in SON mode.

Starting with QCA\_Networking\_2017.SP5.0.3, in Wi-Fi SON networks, multi-SSID and traffic separation support on root and repeater APs that are configured with QCA9886 chipsets and on any platform that uses the QCA9886 radio (for example, IPQ4019 (ARM processor) + QCA9886).

### 19.3.1 High Level Design



### 19.3.2 Setup

- A special AP VAP is needed, which will let only RE Stations to connect.
- Bridging is done on RootAP and RE as shown in the block diagram above.
- Special AP VAP at RootAP will have multiple VLAN interfaces created on top of it.
- STA interfaces at the RE will have multiple VLAN interfaces created on top of them.
- VLAN interfaces created will be associated to appropriate bridge as shown in the block diagram above.
- Private VAPs will be part of private bridge and guest VAPs will be part of Guest Bridge as shown in the block diagram above.

#### 19.3.2.1 Traffic flow

- Packets between Special VAP and RE stations will be VLAN tagged.
- Untagged packets arriving at the two bridges from VAPs (Guest, Private) or from Ethernet interface (RootAP), or from gateway (RootAP) is tagged with appropriate VLAN id before sending the packet to the special AP VAP (RootAP) or STA(RE) if the destination of the packet is not local (included clients associated to guest or private VAPs).
- When a tagged packet arrives at special AP VAP (RootAP) or STA (RE), the packet is untagged and forwarded to appropriate bridge based on VLAN id. If the id is other than guest

or private network id then the packet is dropped. The packet arrived is either consumed locally (included clients associated to guest or private VAPs) or forwarded to the backhaul (WAN interface in case of RootAP).

### 19.3.3 Configuration changes

#### 19.3.3.1 Network configuration changes

##### RootAP

1. Create two VLAN interfaces for Special VAPs athN.1 and athN.2.
2. Attach athN.1 to private bridge and athN.2 to guest bridge. See [Section 19.3.1](#).
3. VLAN interface creation on special VAP for guest network is controlled by user input.

By default VLAN interfaces for guest network is created only on 2.4 GHz special VAP.

VLAN interfaces can be created either on 2.4 GHz or 5 GHz special VAPs but not on both.

This is done in order to avoid looping in guest network.

##### Repeater

1. Create two VLAN interfaces for special STAs athX.1 and athX.2.
2. Attach athX.1 to private bridge and athX.2 to guest bridge.
3. VLAN interface creation on special VAP for guest network is controlled by user input.

By default VLAN interfaces for guest network is created only on 2.4 GHz special VAP.

VLAN interfaces can be created either on 2.4 GHz or 5 GHz special VAPs but not on both.

This is done in order to avoid looping in guest network.

#### 19.3.3.2 Wireless configuration changes

##### RootAP

1. Create guest and special VAPs in addition to existing VAPs on each radio. Let us call the existing VAPs as Private VAPs.
2. Guest VAPs and special VAPs get the same SSID as that of private VAP but with a small modification. Guest VAPs SSID will have a suffix ‘-guest’ and special VAP SSID will have a suffix ‘-spcl’.
3. Guest VAPs are configured with open security whereas special VAPs receives same security as that of Private VAP.
4. By default SSID on special STAVAPs will be hidden.
5. WPS will be disabled on both Guest and special VAPs.

## Repeater

1. Create guest and special VAPs in addition to existing VAPs on each radio. Let us call the existing VAPs as private VAPs.
2. Once connected to private VAP on Root AP (using WPS) special STA acquires SSID and security credentials from private VAP of RootAP.
3. As mentioned earlier Guest VAPs and Special VAPs on Repeater get the same SSID as that of Private VAP of RootAP but with a small modification. Guest VAPs SSID will have a suffix ‘-guest’ and Special VAP SSID will have a suffix ‘-spcl’.
4. Guest VAPs are configured with Open security whereas Special VAPs receives same security as that of Private VAP of RootAP.
5. WPS will be disabled on both Guest and special VAPs.
6. Private STA on Repeater is reconfigured to act as special STA as shown in the diagram.

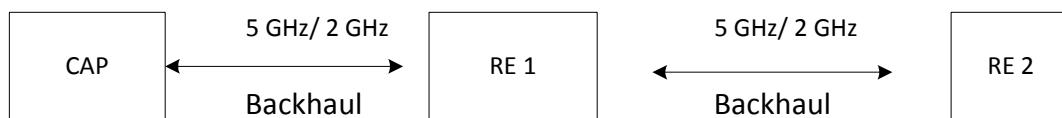
### 19.3.4 Driver changes

- As discussed above special VAP should accept connection request only from Special STA.
- To achieve this, following driver modifications are required:
  - Special STA should include a new IE in association request.
  - Special VAP looks for this IE in association request to continue connection procedure. If this IE is not present in association request then the connection is refused.
  - Also VLAN handling is disabled on the driver. This makes Linux network stack to take care of VLAN handling.

## 19.4 Wi-Fi SON: Full Daisy Chain support across multiple hops

This feature adds full daisy chain support to the Wi-Fi SON feature. Daisy chain allows a RE to connect to another RE, which can be used to extend the range further. Full daisy chain supports multi-hop capability. AP steering and path selection across multiple hops (hop-by-hop decision) is supported.

The following figure shows a simple Daisy chain network, where RE1 is connected to the CAP and RE2 is connected to RE1. Both REs have HYD instances running and AP and Band steering will take places as usual across them.



Full Daisy Chain support implementation currently has the following limitations.

1. SON features like APS is supported from hop-to-hop and not end-to-end i.e. traffic going to CAP to STA connected to RE-2 would not consider end-to-end path during path selection process.
2. Clients beyond RE1, connected to distant neighbor's front-haul will show as bridged devices of RE1 in topology display (td s) and also on multi-hop RE that they are connected to.
3. There is no forced hard-limit to number of hops, but 2 hops is recommended for such set ups.
4. It is recommended to run same software version. For example. non-daisy chain CAP and daisy chain supporting REs kind of set up should be avoided.
5. Ethernet and PLC connections between CAP and Distant neighbor needs to be tested and handled.

The following assumptions are made during this implementation.

1. Best uplink selection algorithm will help build multi-hop (daisy chained) topology when more effective than star topology.
2. RE contains only one upstream parent device. A scenario in which 5G backhaul connects to one RE and 2G backhaul connects to a different RE is not supported.

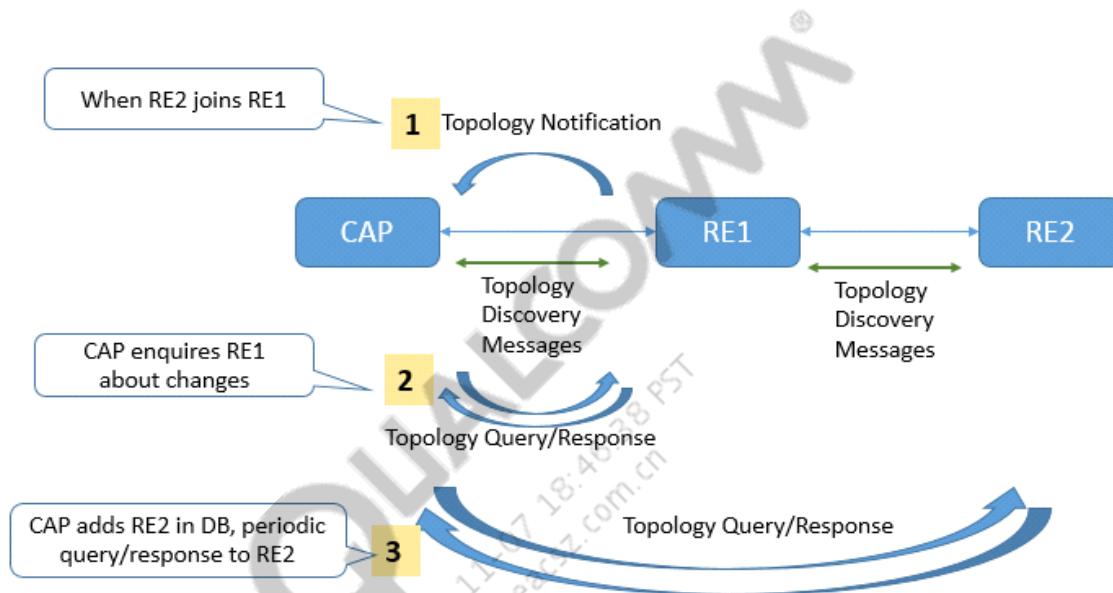
## **Topology building process**

Topology message exchange and processing makes the core of daisy chain support. 1905 Topology messages are used to learn about network topology by all nodes in the network. Following are topology messages:

1. Topology Discovery: Neighbor-Multicast
  - a. Sent by each node on all interfaces periodically
  - b. Consumed by direct neighbor, never forwarded
  - c. Helps build DIRECT NEIGHBORS
2. Topology Notification: Multicast-Relay
  - a. Sent whenever a node detects change on its topology or interface i.e. new link up/down, STA associated/disconnected etc.
  - b. Relayed by all nodes, nodes discard previously seen Notifications to prevent loop
3. Topology Query: Unicast
  - a. Sent by a node to a specific destination to query of its current state and interfaces
  - b. Periodically sent for each entry in Database
4. Topology Response: Unicast
  - a. Sent in response to Topology Query to querying node
  - b. Provides its current state
  - c. Used by both Direct and Distant Neighbors
  - d. Has following TLVs
    - (a) Device Info

- (b) Capabilities
- (c) List of Legacy Neighbors (for each of its interfaces)
- (d) 1905 Device Neighbors on its interfaces
- (e) Attached WLAN Stations

The following figure shows the topology message exchange example when RE2 joins RE1 and how these messages are used to update current topology at CAP.



**Figure 19-6 1905 Topology message exchange example**

### Design changes

Changes to support daisy chain includes following broad items:

1. Adding Distant Neighbor concept. All nodes that are not direct neighbors will fall into Distant Neighbor category irrespective of hops
2. Detect Distant Neighbor and add it in Topology Database
3. Add periodic message (heartbeat/hello) exchange to detect distant neighbor is still there. This is done using Topology Query and Response messages
4. Handle expiry and age-out for Distant neighbor Topology Database entry
5. Extend Topology Database iterator to include Distant Neighbor entries during iteration
6. Include Distant Neighbors in steering decisions and steering messages
7. Extend relationship categories for nodes in daisy chain topology i.e. adding support to detect if another device is in upstream path or downstream path or a peer node.
8. Enhance topology display to include more information like Direct or Distant Neighbor, Upstream device etc. Adding additional modes to display topology (td 1/td s2 in addition to td s)

## Adding Distant Neighbor Device

Topology Response message from a node contains IEEE1905\_TLV\_TYPE\_NEIGHBOR\_DEVICE, which lists 1905 Neighbor devices for each of its interfaces. While processing Topology Response NEIGHBOR\_DEVICE TLV from a direct neighbor, if a particular 1905 Device is not present in processing node's Database, a new DB entry is created and this neighbor is added as DISTANT NEIGHBOR. A topology Query message is immediately scheduled to get more information directly from this Distant Neighbor. Periodic Topology Query message is also scheduled for this Distant Neighbor entry. In addition to DB entry, this Distant Neighbor is also added to Bridged Device List of Direct Neighbor from where this Topology Response was received. Distant Neighbor status could change to Direct Neighbor if processing node ever starts receiving Topology Discovery message for a 1905 device. Code changes in file **qca-hyd/services/tdService/tdService1905.c**, function **tds1905ResponseHandle()**.

Example:

CAP -----RE1 -----RE2

When RE2 joins RE1, RE1 sends Topology Notification to CAP. CAP then sends Topology Query to RE1 and RE1 responds with Topology Response. When CAP is processing RE1's Topology Response, it will find RE2 listed as Neighbor Device. CAP will add RE2 as Distant Neighbor and it will also add RE2 in list of Bridged devices in RE1's DB entry. CAP will send Topology Query to RE2 and receive detailed information about RE2 in RE2's Topology Response message. Periodic Query/Response will happen between CAP and RE2.

## Handle ageout for distant neighbor DB entry

Periodic Query and Response between neighbors (direct or distant) will act like heartbeat or hello. With reception of Topology Response message from a Distant Neighbor, its DB entry is refreshed. Since there is no interface connection with Distant Neighbor, only device level expiry exists for Distant Neighbor. Note that Direct Neighbors have expiry on active and connected interfaces as well. Upon age-out for Distant Neighbor, DB entry is deleted. Changes are in file **qca-hyd/services/tdService/tdServiceDB.c**, function **tdsDBEntryAging()**. New function added **tdsLoopAvoidanceAndDelDBEntry()**.

## Extend relationship between devices

Each call for get relationship returns one of three choices for a target node: upstream, downstream or neighbor. We are keeping same return values, but changing logic inside relationship by storing additional information about devices as topology is created. New functions are added to obtain and store extra information during processing of Topology messages. New parameters **addrBSSID** and **upstreamWifiDevice** stores BSSID for a Wifi Interface and the upstream 1905 device for a given device respectively. upstreamWifiDevice will be NULL for CAP. This information is obtained and updated while creating and processing Topology Response messages. Topology Response from a device contains Device Info TLV, which carries detailed information about each Interface Type, Role, BSSID etc. If an interface is in WDS STA role, it will carry BSSID info to identify which WDS AP it is connected. This BSSID information is stored in addrBSSID. This addrBSSID information is then used to find out which 1905 device it belongs to and that 1905 device will be stored as upstreamWifiDevice for the Topology Response's source device. If Interface is in WDS AP role, BSSID will be of its own. addrBSSID will be updated but upstream device search will be skipped.

During relationship computation between self and target device, upstreamWifiDevice is used to find out if target node is either in upstream direction or downstream direction. If neither, they are considered neighbor nodes. Relationship function also adds number of hops between self and target device as return value, which can be used to influence steering decision. Code changes in **tdService\_DBGetRelationship()**.

New functions:

```
tdssDBUpdateLocalInterfaceRoleAndBSSID(),
tdsDBUpdateUpstreamWifiDevice(),
tdsDBGetUpstreamWifiDevMac(),
tdsDBGetUpstreamDevice(),
tdsDBIsUpstream()
```

### **Include distant neighbor in Topology Database iteration**

Topology database iterators are extended to include Distant Neighbors. In **services/tdService/tdServiceDB.c**

Two new functions to iterate through DISTANT NEIGHBORS as well.

```
tdDeviceHandle_t tdService_DBGetFirstHandleExt()
tdDeviceHandle_t tdService_DBGetNextHandleExt()
```

### **Include Distant Neighbors in Steering decisions/messages**

The wlb functions are updated to include distant neighbors on steering messages and decisions. Use extension functions for iterating DB. Use **tdService\_DBGetFirstHandleExt()**, **tdService\_DBGetNextHandleExt()** in following functions in wlb:

```
wlb/bandmon/bandmonMBSA.c
bandmonMBSAFindNextAllowedAP()
bandmonMBSAResetLoadBalancingAllowedAP()
wlb/steerexec/steerexecImplMBSA.c
steerexecImplMBSAShouldSendMsg()
wlb/steermsg/steermsg.c
steermsgSendMsgToAllNodes()
```

### **Debug messages added for enhancing topology display**

Extra information is stored during Topology creation and updates. This additional information is used to provide more information about connectivity and hops in topology display—td s. In addition to td s, two more levels of topology display added:

- td s1
- td s2

Code changes in **tdsPrintTDDBEntry()**.

New functions added:

```
tdsIsBSSIDSetForRole(),
relationsToOtherDevices(),
tdServiceMenuStatus1Handler(),
tdServiceMenuStatus2Handler()
```

## Displaying Locally connected Wifi Clients

Currently, all Destination Addresses beyond first hop are displayed only as Bridged addresses on direct neighbor. It is difficult to find if Wifi Client is connected to RE1 or RE2 or beyond. In order to fix this, a new function is added **tds1905CopyAssociatedWifiDevices()** to add List of Associated Wifi STAs in VENDOR\_SPECIFIC TLV to list of Bridged Devices. This function is called only for Distant Neighbor, while processing VENDOR SPECIFIC TLV for Associated STAs in Topology Response message.

## Guidelines

Keep the following points in mind while employing the Wi-Fi SON software version with full daisy chain support:

1. Daisy Chain (full) is not a configuration parameter, unlike Daisy Chain Lite. In other words, it cannot be enabled and disabled via configuration. It is always present. New software should not change anything for star topology that was working before. It just adds capability to support DAISY CHAIN topology as well.
2. REs in star topology are all Direct Neighbors. In DAISY chain, one would see Distant Neighbors as well in Topology display.
3. All REs will now have two VAP on each backhaul radio, WDS STA VAP to connect to upstream device and WDS AP VAP to provide coverage to another RE if it comes up later.

For example: CAP – RE1 – RE2, here RE1 is connected to CAP via 5G WDS STA VAP and connected to RE2 via 5G WDS AP VAP.

4. When a Wifi Client Station is connected to Distant Neighbor's front-haul, it will show up in two places in Topology display when seen at CAP: once under Bridged device of CAP's direct neighbor and then again under actual Distant Neighbor entry.

For example: CAP – RE1 – RE2, STA2 connected to RE2

During td s at CAP, STA2 will show up as Bridged device in RE1's DB entry and Bridged Device at RE2's DB entry. In such cases, whenever Wifi Client shows up at Distant Neighbor DB entry, always consider that Wifi Client is connected to that Distant Neighbor (and not Direct neighbor).

5. For above scenario (#4), hmwds table at CAP will RE1 as next hop for all addresses beyond RE1. In example scenario above (#4), even though STA2 is connected to RE2, at CAP, hmwds will show RE1 as next hop. RE1 will also have hmwds table for STA2 and that will point to RE2.
6. HYD should be running on all REs and AP steering should happen as expected between REs.
7. It is easier to create daisy chain if RE2 comes up after RE1.

## Sample configuration scenarios

This subsection describes examples of configuration environments with full Wi-Fi SON daisy chain support. These scenarios use DK07 (three radio AP) and with one radio (5G QCA9984) for dedicated backhaul.

## Single CAP –RE topology

Simple topology test for basic Wi-Fi SON working. CAP connected to a RE, where both devices were running new software. Check was done to see if RE connects to CAP and topology is getting created as expected. “td s” display and “Topology message exchange” was verified via HYD logs. FDB and HD tables were also verified and Ping traffic was used to check connectivity.

## Simple CAP – RE, New version CAP, older version RE

This sample scenario comprises a simple CAP – RE topology with new software on CAP, older software of RE was used to determine if version mismatch will affect existing network. Version mismatch does not cause any errors or malfunctions in existing star topology. Older version devices will not understand DISTANT neighbor concept, so they will only form star and will not allow any DAISY CHAINING through them. Connection, data flow, steering etc. will still happen as in star topology.

## Simple star topology RE1 – CAP – RE2

This sample scenario comprises a simple star topology check if new software does not break previous basic star topology features. “td s”, “FDB and HD” tables were verified. Ping and AP steering work correctly and no anomaly is observed.

## Simple DAISY CHAIN topology CAP – RE1 – RE2

This sample scenario comprises a simple two hop Daisy Chain topology. First CAP and RE1 were brought up and then RE2 was booted up in a placement where RE1 was closer to RE2 than CAP. RE2 picked RE1 to reach CAP, thereby forming Daisy Chain topology. “td s” and “FDB, HD, hmwds tables” were verified for DIRECT and DISTANT Neighbor concept. “td s1” and “td s2” was also verified which provides extra information about connections. It was also verified that “average Utilization reports” were being sent by all nodes (as seen in HYD logs). Ping tests were also done with iPhone-6 as client. NOTE: Wifi Client will be displayed twice in “td s” when connected to Distant Neighbor.

## Daisy chain with RE1 reboot

In existing Daisy Chain topology CAP – RE1 – RE2, RE1 was rebooted to examine whether network recovers correctly. RE2 was able to join CAP directly and “td s” reflected this change by converting RE2 entry at CAP to “DIRECT” neighbor from “DISTANT” neighbor. FDB, HD and hmwds tables were updated accordingly. When RE1 came up, it also joined CAP as DIRECT neighbor.

When RE1 came back up, both RE1 and RE2 remained as DIRECT neighbor and existed as star topology. It did not convert back to DAISY chain topology. Best uplink selection algorithm should periodically check if better uplink is available.

## 19.5 Wi-Fi SON: Support across multiple Hy-Fi bridges

Multiple HYD instances or multiple BSSIDs (one in the public network and another in the private network) supporting AP steering and path selection in the public and private networks are supported. This section describes how the existing Hy-Fi bridging module, with a single bridge support, is modified to support multiple Hy-Fi bridges.

Until QCA\_Networking\_2016.SPF.3.0 release, Hy-Fi bridging module supports one bridge only. For customers using multiple SSIDs and multiple VLANs, which involves multiple bridges, Wi-Fi SON was supported only on one bridge. Starting with QCA\_Networking\_2016.SPF.4.0 release, Wi-Fi SON is supported across multiple Hy-Fi bridges.

The following are the high level changes to the HyFi data structures.

The old ‘hyfi\_br’ Structure definition is static struct hyfi\_net\_bridge hyfi\_br \_\_read\_mostly;

The new ‘hyfi\_br’ Structure definition is struct hyfi\_net\_bridge hyfi\_br[HYFI\_BRIDGE\_MAX] \_\_read\_mostly;

A new macro is introduced to define Number of bridges. This can be changed at compile time.

```
#define HYFI_BRIDGE_MAX 2 /* max number of hyfi bridges supported */
```

New members to *struct hyfi\_net\_bridge*, to store the Linux bridge name are

```
char linux_bridge[IFNAMSIZ]; /* Linux bridge name */
```

New members to *struct net\_hatbl\_entry* and *struct net\_hdtbl\_entry* are

```
struct hyfi_net_bridge * hyfi_br;
```

This is the pointer to the HyFi Bridge instance to which the HA/HD Table entry belongs to.

Removal of `#define HYFI_BRIDGE_ME (NULL)` is done.

All code using HYFI\_BRIDGE\_ME determines the HyFi bridge instance that it requires using the device interface, bridge interface, or skb.

### Hy-Fi Functional Changes

The following are the new functions:

- Function to get the HyFi Bridge instance by using the Linux Bridge name.

```
struct hyfi_net_bridge * hyfi_bridge_get_hyfi_bridge(const char *br_name);
```

- Function to allocate a HyFi Bridge instance from the ‘hyfi\_br’ array.

```
struct hyfi_net_bridge * hyfi_bridge_alloc_hyfi_bridge(const char *br_name);
```

- Function to get HyFi Bridge instance by using the Linux Bridge port instance.

```
struct hyfi_net_bridge *hyfi_bridge_get_by_port(const struct net_bridge_port *port);
```

The following existing functions identify the instance of the HyFi Bridge that is being used. So these functions will have to support the addition of an extra point to ‘*struct hyfi\_net\_bridge*’. The new definitions of these functions are as follows:

```

void mc_nbp_change(struct hyfi_net_bridge *hyfi_br, struct net_bridge_
port *p, int event);
void mc_fdb_change(struct hyfi_net_bridge *hyfi_br, __u8 *mac, int
change);
static int hyfi_bridge_ports_init(struct hyfi_net_bridge *hyfi_br, struct
net_device *br_dev);
static int hyfi_bridge_init_bridge_device(struct hyfi_net_bridge *hyfi_
br, const char *br_name);
static int hyfi_bridge_deinit_bridge_device(struct hyfi_net_bridge *hyfi_
br);
static int hyfi_bridge_del_ports(struct hyfi_net_bridge *hyfi_br);
int hyfi_bridge_dev_event(struct hyfi_net_bridge *hyfi_br, unsigned long
event, struct net_device *dev);
int hyfi_bridge_set_bridge_name(struct hyfi_net_bridge *hyfi_br, const
char *br_name);
int hyfi_bridge_init_port(struct hyfi_net_bridge *hyfi_br, struct net_
bridge_port *p);
static struct net_bridge_port *hyfi_bridge_get_dst_port(struct hyfi_net_
bridge * hyfi_br, const struct net_bridge *br, u_int32_t hash, u_int32_t
traffic_class, u_int32_t priority, struct sk_buff *skb, const unsigned
char *dest_addr, const unsigned char *src_addr, struct net_hatbl_entry
**ha_out)
void hyfi_netlink_event_send(struct hyfi_net_bridge *br, u32 event_type,
u32 event_len, void *event_data);
void hyfi_netlink_event_send(struct hyfi_net_bridge *hyfi_br, u32 event_
type, u32 event_len, void *event_data);

```

## HA/HD Table Changes

The HyFi HA and HD Tables will have the ‘*hyfi\_br*’ pointer initialized to the instance of the HyFi Bridge they belong to. This initialization is done in the functions *hatbl\_create* and *hdtbl\_create*.

Following HA/HD related functions can use this pointer for any HyFi Bridge structure access.

- *hyfi\_hatbl\_insert*
- *hyfi\_hatbl\_insert\_ecm\_classifier*
- *hyfi\_hatbl\_insert\_from\_fdb*
- *hyfi\_bridge\_handle\_ha*

## Bridge/Interface Event Notification Changes

The function *hyfi\_device\_event*, which is the registered notification function, needs to derive the HyFi Bridge instance of the current network device. This is done as shown in the below code snippet.

```

static int hyfi_device_event(struct notifier_block *unused, unsigned long
event, void *ptr)
{
    struct net_device *dev = ptr;
    struct net_bridge_port *p = hyfi_br_port_get(dev);
    struct hyfi_net_bridge *hyfi_br = hyfi_bridge_get_by_dev(dev);

```

This instance of HyFi Bridge is passed to functions like `hyfi_bridge_dev_event` and `hyfi_netlink_event_send` for further action.

The function `hyfi_br_notify`, which is a hook to the Bridge notifications, also needs to derive the HyFi bridge instance. This is done as shown in the below code snippet.

```
void hyfi_br_notify(int group, int event, const void *ptr)
{
    struct net_bridge_port *p = (struct net_bridge_port *)ptr;
    if (p)
        hf_br = hyfi_bridge_get(p->br);
```

This instance of `hyfi_br` is passed to functions like `hyfi_bridge_init_port` and `hyfi_bridge_delete_port` etc....

## Netlink Support Changes

The `hyfi_netlink_receive` function will use the Linux Bridge name in the netlink message to derive the HyFi Bridge instance. The ‘hyd’ or ‘hyctl’ application is expected to set the correct Bridge name, while sending the Netlink message. This is done as shown below.

```
static void hyfi_netlink_receive(struct sk_buff * __skb)
{
    struct net_device *brdev = NULL;
    struct hyfi_net_bridge *hyfi_br;
    if ((skb = skb_clone(__skb, GFP_KERNEL)) != NULL)
        nlh = nlmsg_hdr(skb);
    hymsg hdr = NLMSG_DATA( nlh );
    brdev = dev_get_by_name(&init_net, hymsg(hdr)->if_name);
    hyfi_br = hyfi_bridge_get(netdev_priv(brdev));
```

The Netlink Attach message allocates the instance of HyFi Bridge using the function `hyfi_bridge_alloc_hyfi_bridge`. This instance is passed to Function `hyfi_bridge_set_bridge_name`, along with the Linux Bridge name for attaching the HyFi Bridge with the Linux Bridge.

The Detach message calls function `hyfi_bridge_set_bridge_name` with a NULL Bridge name to detach the HyFi bridge from the Linux Bridge.

There is not much change to the `hyfi_netlink_event_send` function, except for the addition of the new argument (`struct hyfi_net_bridge *`). All calling functions should pass this instance of HyFi Bridge. The ‘event\_pid’ member of the HyFi bridge structure is used to send the Netlink message to the correct instance of the hyd application.

## 19.6 Save STA information to flash for Wi-Fi SON

Hy-Fi daemon (Hyd) maintains info regarding associated stations in a database called stadb. Stadb is not a shared database, meaning a completely independent copy of stadb exists within the hyd process, and further, other users of the stadb shared libraries (lbd for instance) have their own in-process copy of stadb. Multiple instances of stadb do not share data.

Because stadb is not a shared database, all data stored within the database is lost when the controlling process stops. The lifecycle of the data is the same as that of the process.

A capability to save STA information into flash memory is introduced. The functionality enables addition of add persistence capabilities to hyd or to stadb such that the data stored within stadb (steerexec info in particular) is saved onto flash before the hyd process is stopped, and subsequently restored into stadb when hyd is started. This mechanism should be able to persist information across hyd restarts and across system reboots.

For stadb, qca-whc-lbd/libs/stadb/ contains the database code. It implements a hashtable keyed by MAC address and stores a pointer to memory provided by the controlling process.

For hyd, qca-hyd/ is the controlling process and allocates memory for STA data. In addition, qca-hyd saves a pointer to this memory in stadb, keyed by MAC.

Persist algorithm works as follows:

- Cycles through all devices in stadb
- For each device, converts struct data into JSON format and performs the following:
  - Open tmp file (filename~)
  - Write file
  - Close tmp file
  - Move tmp file to final persistence file location
- The triggers are timer-driven (currently 5 seconds) or initiated only when the file is uclean.

Restore algorithm works as follows:

- Open persistence file
- Read JSON
- Close persistence file
- Cycle through entries in JSON
- For each device entry
  - Convert JSON object into device fields
  - Add device MAC
  - For each field, set the corresponding field in the device data
- The trigger is the start of the hyd process.

The following is an example of JSON-formatted data that is written to a persistence file.

{

```

    "version": 1,
    "devices": {
        "00:11:22:33:44": {
            "uplinkinfo": { "lastUpdateSecs": <number>, ... }
            "downlinkinfo": { ... },
            "bss": { ... },
            "lastUpdateSecs": <number>,
            "pollutionExpirySecs": <number>,
            "reservedAirTime": <number>,
            "valid": <string>,
            "polluted": <string>,
            "steerexec": { ... },
            "steermmsgState": {... },
        },
        "00:22:44:66:88": { ... },
        ...
    }
}

```

The following fields are persisted:

- “steerexec” info.
- inNetworkInfo
- outNetworkInfo.

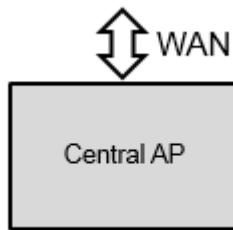
## 19.7 Detection of WAN and LAN sample scenarios

This section describes the different deployment environments in which detection of WAN and LAN links is supported. These configuration scenarios apply only for IPQ4019 chipsets. Before detection, the initial state of the network interfaces is set. Link Layer Discovery Protocol (LLDP) is used to determine the furthest upstream Wi-Fi SON device. LLDP daemon is running on each Wi-Fi SON device. The reconfiguration as CAP/RE is not performed here. The starting state for any device is CAP mode. Interfaces are identified as upstream or downstream by the bridge loop prevention code before running LAN/WAN detection. Although both star and daisy chain topologies can be supported, only upstream or downstream detector is used from the bridge loop prevention code. As a result, currently, support is provided for star topology only. Only one WAN connection is present.

The WAN and LAN autodetection code is disabled by default. Set the following configuration in UCI before starting repacd:

```
uci set repacd.EnableEthernetMonitoring=1
```

## Sample scenario 1



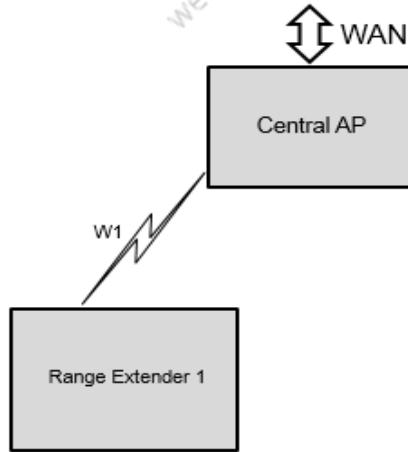
### Initial State

The CAP has no network links at all

### Steps

- An Ethernet cable is plugged into the CPE and CAP Ethernet port n.
- The CAP obtains a DHCP address.
- Poll LLDP daemon for 30 seconds waiting to see if any other Wi-Fi SON devices are visible on upstream interfaces (port n).
- The LLDP daemon poll times out after 30 seconds (configurable) so that it is identified that port n is a WAN interface.

## Sample scenario 2



### Initial State

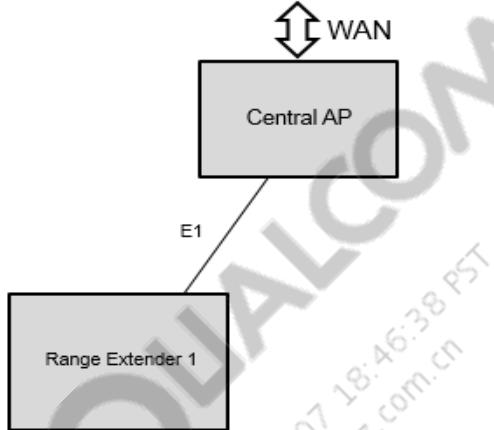
The CAP already has a WAN connection

RE1 is configured as a CAP

### Steps

- W1 interface comes up
- On RE1, W1 is identified as upstream and gets a DHCP address.
- On, RE1, we poll LLDP daemon for 30 seconds (configurable) looking for any other Wi-Fi SON devices on upstream interfaces (W1).
- It is seen that the CAP is an upstream Wi-Fi SON device to identify that W1 is a LAN interface.

### Sample scenario 3



#### Initial State

The CAP already has a WAN connection

RE1 is configured as a CAP

#### Steps

- E1 interface comes up
- On RE1, E1 is identified as upstream and obtains a DHCP address.
- On, RE1, we poll LLDP daemon for 30 seconds (configurable) looking for any other Wi-Fi SON devices on upstream interfaces (E1).
- It is seen that the CAP is an upstream Wi-Fi SON device so that it is determined that E1 is a LAN interface.
- The LLDP profile contains the following specifications:
  - Memory: VSZ 1032 KBytes (using ps)
  - Disk: 290 KBytes (using ls -l)
  - Processor: approximately 0 % (through the top)

## CAP Multi-WAN Prevention

The prerequisite is that only one WAN connection Wi-Fi SON device exists in the Wi-Fi SON network with a WAN connection.

On a single Wi-Fi SON CAP, a user may plug in two or more WAN Ethernet cables. We must detect the condition and disable all but a single WAN port in order to automatically enforce the condition that one and only one WAN interface is enabled and available in a Wi-Fi SON network.

1. Step 1 of this feature entails the detection of multiple WAN links. Each WAN link on a CAP will be identified as such so that appropriate action may be taken by a controller process.
2. Step 2 of this feature implements the action of downing all WAN interfaces except the first available port.

## Limitations

This feature does not cover any form of failover or load balancing of the WAN links. In other words, if multiple WAN links are plugged in to a single device and all WAN links except the first one has been disabled, and then the one remaining WAN port loses internet connectivity, this feature will not enable one of the previously disabled WAN ports.

## CAP Bridge/Router Autodetection

This feature introduces a new operating mode called CAP Bridge. The following are the characteristics of a CAP Bridge:

- CAP, but configured for pure bridge
- Not a CAP because NAT/firewall is disabled.
- Not a CAP because DHCP server is disabled.
- Not a RE because Wi-Fi STA interfaces are disabled.
- Not a RE because it has the WAN uplink.
- Not a RE because it is the furthest upstream Wi-Fi SON device.
- Not a RE because upstream device is not Wi-Fi SON, which means support for

hyfi-non-relaying port group is unknown. This is the first case where we could potentially have a hyfi non-relaying port group connected to a non Wi-Fi SON device upstream.

The purpose of the feature is to detect when a device should automatically reconfigure from CAP router to a CAP bridge and to perform the reconfiguration.

1. Step 1 of this feature is detection. Determine the conditions which necessitate reconfiguration. Run reconfiguration when a firewall is detected upstream.
2. Step 2 of this feature is reconfiguration. Implement reconfiguration script which converts a CAP router into a CAP bridge.

```
uci set dhcp.lan.ignore=1; uci commit dhcp;
/etc/init.d/dnsmasq restart
uci set network.lan.ifname='eth0 eth1';
uci set network.lan.proto=dhcp
uci delete network.wan; uci commit network
```

```
/etc/init.d/network restart
```

If you want to use static IP for ‘br-lan’ change uci set network.lan.proto=**dhcp** to uci set network.lan.proto=**static**

## Limitations

This feature covers reconfiguration from CAP Router to CAP Bridge and not the reverse.

## 19.8 Network loop prevention sample scenarios

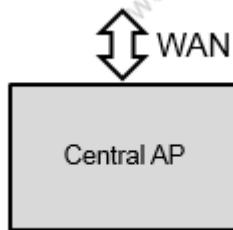
This section describes the different deployment topologies in which network loop prevention is supported. For the discussion of this feature, the following definitions are established. WAN is defined as the single connection to the Internet (cable modem, DSL modem). All other connections are considered as LAN. There can only be one WAN connection (uplink to CPE). Only star topology is supported; daisy chain topology is not supported.

The WAN and LAN autodetection code is disabled by default. Set the following configuration in UCI before starting repacd:

```
uci set repacd.EnableEthernetMonitoring=1
```

Upstream is a connection to CAP in a star topology. Downstream is a connection from the CAP to an RE or a connection from an RE to a client. Hy-Fi is enabled in all these configuration scenarios. The devices have already been configured as a CAP or an RE.

### Sample scenario 1



### Prerequisites

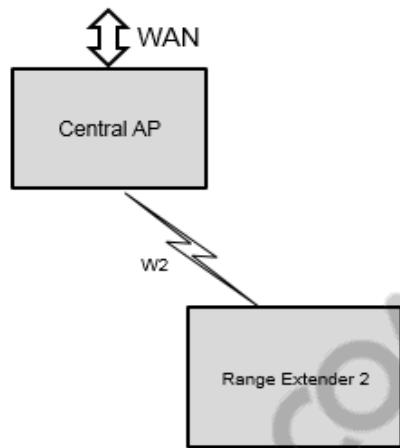
- The CAP has no network links at all
- Each Ethernet port is in its own VLAN with its own virtual interface.
- br-lan has the Wi-Fi interfaces in it but no Ethernet interfaces.

### Steps

- An Ethernet cable is plugged into the CPE and CAP Ethernet port 1.
- A DHCP request is sent on eth1.1.
- A DHCP response comes back from the CPE so we classify eth1.1 as an upstream port
- eth1.1 is configured as the wan network.

- All other interfaces are added to br-lan

## Sample scenario 2



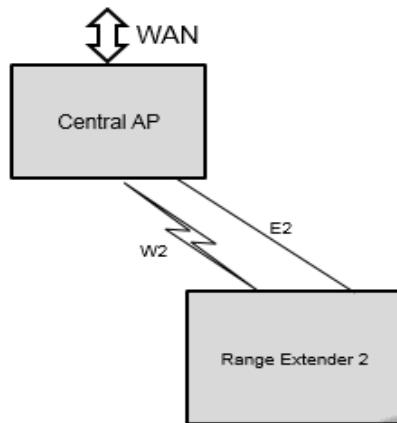
### Prerequisites

- The CAP is already connected to the WAN
- An RE is powered on but not connected.
- On the RE, br-lan is empty.

### Steps

- The WPS button is pressed on both the CAP and the RE
- W2 link comes up
- On the RE, W2 is marked “upstream” since it’s a managed(station) interface.
- The RE adds W2’s interface to br-lan and configures it as non-relaying Hy-Fi.

### Sample scenario 3



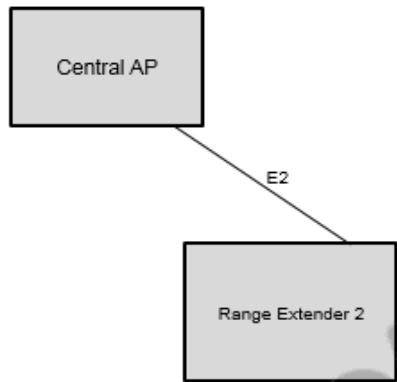
#### Prerequisites

- The CAP is up and connected to the WAN
- W2 is up and configured for Hy-Fi(relaying at the CAP, non-relaying at the RE)
- On the RE, W2 is in br-lan and all Ethernet ports are in their own VLANs with their own virtual Linux interfaces.

#### Steps

- Ethernet link E2 is brought up
- A DHCP request is sent from the RE on E2
- The CAP replies with a DHCP response.
- On the RE, E2 is marked “upstream”
- E2 is added to br-lan and set as non-relaying in Hy-Fi.

## Sample scenario 4



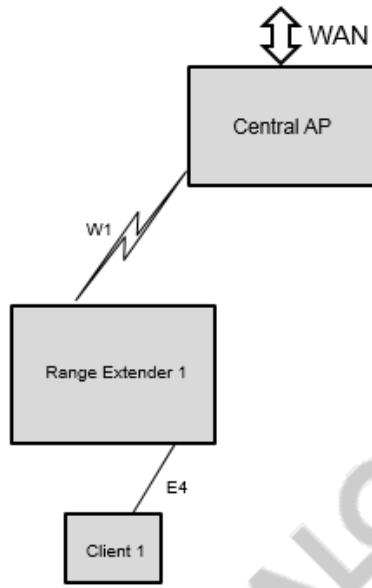
### Prerequisites

- The CAP is up but has no connections. All Ethernet interfaces are in their own VLANs with their own virtual Linux interfaces
- The RE is up but has no connections. All Ethernet interfaces are in their own VLANs with their own virtual Linux interfaces

### Steps

- Link E2 comes up
- The CAP does a DHCP request with a timeout of 15 seconds
- The RE does a DHCP request with a timeout of 30 seconds
- After 15 seconds the CAP gives up and marks E2 as “downstream” and adds it to br-lan.
- RE’s DHCP request succeeds. On the RE, E2 is marked as “upstream”.
- On the RE, E2 gets added to br-lan and set as Hy-Fi non-relaying.

## Sample scenario 5



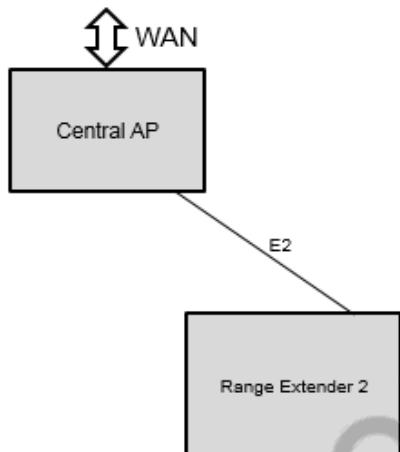
### Prerequisites

- The CAP is connected to the WAN.
- The RE is connected to the CAP via W1.

### Steps

- Link E4 comes up
- The RE performs a DHCP request on E4 with a timeout of 30 seconds
- After 30 seconds, RE marks E2 as “downstream”
- RE puts E2’s port in br-lan and sets it as relaying in Hy-Fi.

## Sample scenario 6



### Prerequisites

- The CAP is up and has a WAN connection. All Ethernet interfaces are in the lan VLAN and are in br-lan.
- The RE is up but has no connections. All Ethernet interfaces are in their own VLANs with their own virtual Linux interfaces

### Steps

- Link E2 comes up
- The RE does a DHCP request with a timeout of 30 seconds
- RE's DHCP request succeeds. On the RE, E2 is marked as "upstream".
- On the RE, E2 gets added to br-lan and set as Hy-Fi non-relaying.

## 19.9 Wi-Fi SON: Temporarily disable access interface

A mechanism to retain the access interface in disabled state when individual VAPs are disabled is implemented. Until QCA\_Networking\_2016.SPF.3.0 release, when a VAP is brought down using the ifconfig mechanism, Wi-Fi-SON daemon brings the VAP up automatically. To avoid the interface coming back up when the user made it administratively down, the interface state is not modified during the creation of the socket. When a user moves an interface to the administratively down state, it is retained in the disabled state. This behavior is achieved during socket creation in libieee1905 by not configuring the interface (which is made down) with IFF\_UP interface Flag when a socket ioctl call with SIOCSIFFLAGS (setflag-0x8914) is made on respective interface.

## 19.10 Enhanced WPS for RE to establish PBC connection in Wi-Fi SON

In the WIFI-SON network with multiple nodes, an enhancement to the WPS mechanism is implemented for the push-button event from Root AP to be propagated to one of the VAP on the Node in the SON domain so RE is able to establish the PBC connection.

This functionality requires WPS PBC to be always performed in Root AP WPS and propagated to other node. It is applicable only for routers in backhaul VAP. When WPS PBC is pushed in ROOTAP, Action frame is transmitted to the Range Extender (RE) AP's to listen for probe Request with WPS IE with PBC enabled from connecting RE. WPS is enabled in the non-backhaul AP's until WPS is started in backhaul VAP.

After WPS with PBC in probe request is received in RE from the new RE, action frame from RE to Root AP is transmitted to notify the presence of WPS client. The root AP cancels the WPS in all radios and transmits action frames to the first RE from which the action frame was received to start WPS. Immediately after the Action frame to start WPS is received in RE, WPS PBC event is triggered based on the action frame received.

The following five types of action Frames are used for the implementation:

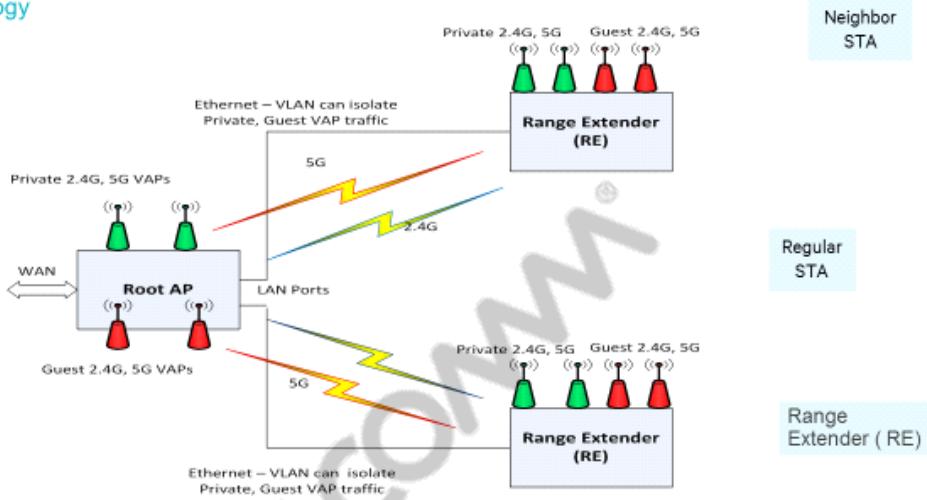
- Action frame from Root AP to RE's to listen for WPS IE in probe request. (WPS\_LISTEN)
- Action frame from RE's to RootAP to inform the WPS client. (WPS\_INFORM\_ROOT)
- Action frame from Root AP to RE to start the WPS.(START\_WPS)
- Action frame from RootAP to all REs to disable wps once wps is already started at Root AP(DISABLE\_WPS)
- Action frame from RootAP to particular RE (when RE informs of wps ie in probe request) to disable wps once wps is already started at one of the REs (STOP\_WPS)

Backhaul must be set in uci for RootAP and RE (both AP and STA vap) as follows:

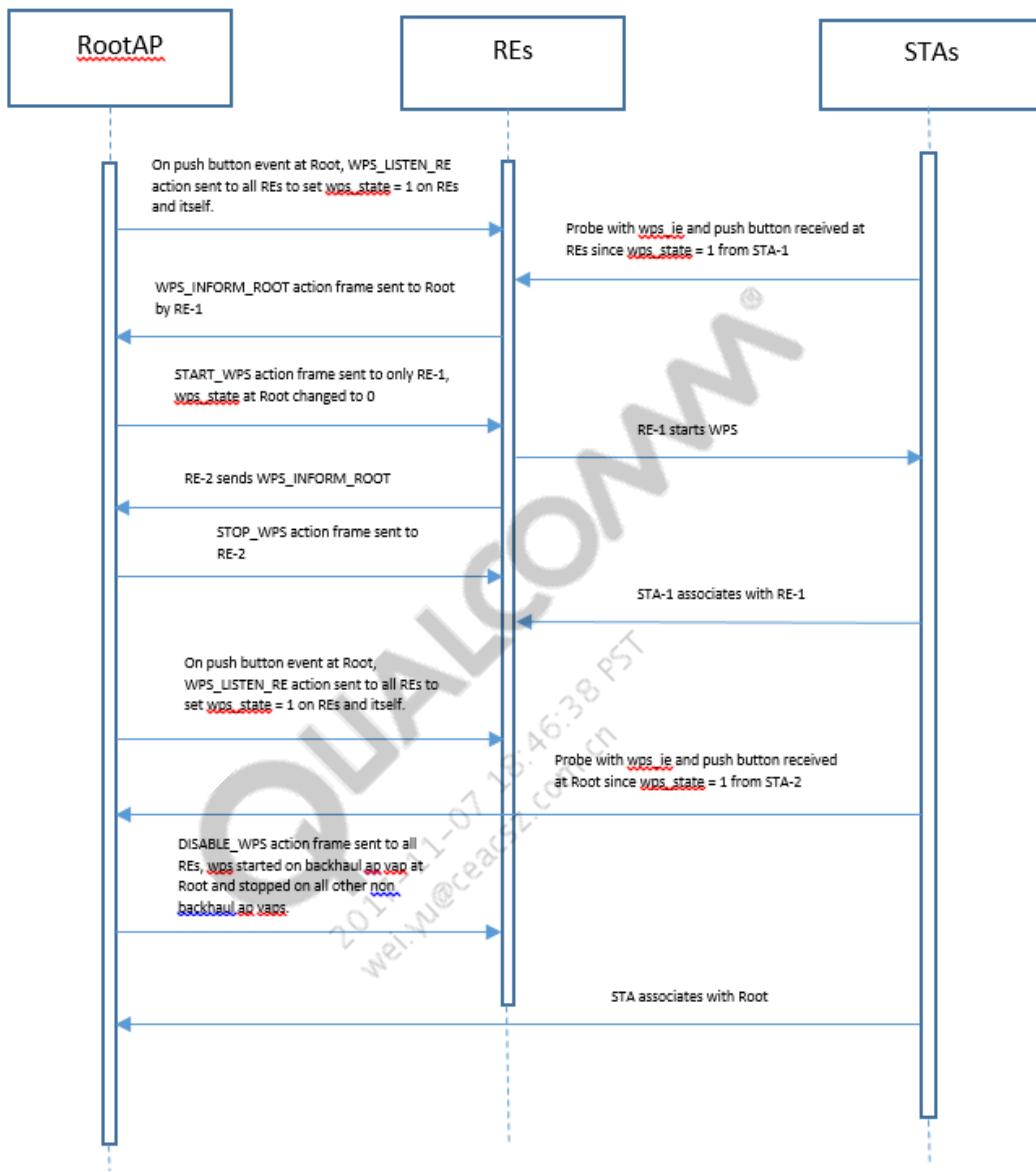
```
uci set wireless.@wifi-iface[0].backhaul=1
```

## WiFi SON Use cases

### Star Topology


**NOTE**

Connection to backhaul or fronthaul is based on vendor-specific IE. Currently, without the check for the vendor-specific IE, clients and REs connect to backhaul VAP. If client initiates WPS on fronthaul, the backhaul VAP cancels it. The functionality for enhanced WPS for RE to establish PBC connection in Wi-Fi SON (`uci set wireless.@wifi-iface[ 0 ].backhaul=1`) must be configured only if additional check is present for vendor-specific IE.



The following sequence of events occur after a WPS PBC event at the root AP in Wi-Fi SON:

- On push button action at the root, the event has to be communicated to the application. This notifies the application to change wps state to start listening for wps ie in probe requests at the root and sends an action frame to its REs.
- On push button at root, action frame is sent to all REs. On receiving this action frame, they change their wps state to listen.
- The REs should be able to send an event to son on receiving action. Son on receiving this action checks type (wps\_listen) and changes wps state.
- The application should see probe request events only if wps push button ie is present in them.
- If root sees a probe request first, it sends a disable wps action frame and immediately starts wps at root and changes wps state to disabled.

- After starting WPS at root AP (on seeing probe events at root), the disable action-frame is sent to REs.
- If RE sees a probe request event and its WPS state is enabled, it sends an action frame to inform the root. The root if enabled (button pushed) sends back a reaction frame.
- After the root receives the above action frame, it frames a reaction frame and forwards this action frame only to the particular RE. Also, the root AP sends the WPS-listen-disable action to all REs and disables WPS listening on it.
- On receiving the start\_wps action from the root, the RE starts WPS and the STA must connect.
- Only root AP receives probe request events and start WPS.
- Only REs receive probe request events and start WPS.
- Either the root or the RE sees probe request event first and connect with STA accordingly.
- STA probe requests are received by root and it connects with the root AP.
- The RE will receive probe request events from the STA and send info to root. Root will send back action frame to this STA and ask to start wps at RE. Disable action frame sent to all REs after wps start at RE.
- REs and the root APs receive probe request events. Depending on the device that receives the request first, STA will associate.
- STA should connect to either of the REs depending on who receives probe first. The other RE should disable listen when wps starts on the other RE.

## 19.11 Fronthaul changes do not restart backhaul in Wi-Fi SON

In a Wi-Fi SON solution (mesh consists of root AP and repeater), cloning of profiles is performed after association of repeater to root AP. Due to profile changes for repeater, several restarts for Wi-Fi subsystem occur. Typically in multi radio solution, a change in parameters for one radio causes a restart for all radios. Such restarts can cause undesired results such as backhaul (STA link from repeater to root) blackout if you change SSID on fronthaul (access VAP) on other radios.

Parameters are divided per underlining radios. If parameters belong to a particular radio, it is not necessary to reset other radios. Certain problems continue to exist, such as fonthaul and backhaul on same radio that change parameters for one of the VAP can reset other VAPs because the underlying radio is the same. However, flows over other radios are not disrupted.

The following assumptions apply:

- Radio parameters except channel, htmode, hwmode, and athnewind continue to reset of all radios presently operating.
- For channel, htmode, hwmode and athnewind, restart is optimized, which tears down all VAPs on present radio and saves Wi-Fi restart.
- Most common VAP parameters such as SSID and encryption do not cause a restart of all radios in system. Only the radios associated with the changed VAP parameters undergo a restart.

## 19.12 Wi-Fi SON: Channel planning on 2.4 and 5 GHz bands for distributed Wi-Fi systems

A mechanism to perform channel planning on 2.4 GHz and 5 GHz access links in a distributed manner and root AP controlled manner is implemented. This capability works in both daisy chain and star topologies.

This functionality enables the setting of the in-network information to driver to enable customers to use this information to perform their own automatic channel selection (ACS) verification. If 5 GHz backhaul quality is good, then 2.4 GHz backhaul is disabled and ACS is performed for 2.4 GHz fronthaul. If 5 GHz backhaul quality is degraded or reduced, then 2.4 GHz backhaul is reconnected. This functionality is supported only for Wi-Fi SON mode. Other repeater modes such as WDS a combination of WDS and EIR, external AP, and QWRAP are not supported.

Use the `set repacd.WiFiLink.independent_channel_selection_enable=1` command to enable the functionality to perform RSSI quality-level checking for independent channel selection. Use the `set repacd.WiFiLink.independent_channel_selection_enable=0` command to disable this functionality.

Use the `set repacd.WiFiLink.independent_channel_selection_rssi_level = -70` command to specify the RSSI value to be considered as the RSSI quality for independent channel selection.

Use the `set repacd.WiFiLink.independent_channel_selection_total_rssi_counter = 10` command to specify the number of iterations or times for which the RSSI-level checking must be performed.

- If RSSI values are lower or higher than -70 for 10 iterations, then quality is good or bad respectively.
- If RSSI value is not lower or higher than -70 for one iteration or cycle, reset the counter and recheck. This mechanism is adopted because using the average of the RSSI values is not effective in multipath environments in which the RSSI value might contain a large difference.

Use the `uci set repacd.WiFiLink.independent_channel_selection_rssi_debug=<0 | 1>` command to enable or disable debugging for RSSI-level checking. 1 indicates that the debug for RSSI level check is enabled, whereas 0 indicates that the debug for RSSI level check is disabled.

The following parameters are added to the `repacd.WiFiLink` setting:

- `is_independent_channel_selection_enable`—This parameter obtains the value from the `repacd.WiFiLink.independent_channel_selection_enable` UCI command. 0 denotes that independent channel selection is disabled, whereas 1 denotes that independent channel selection is enabled.
- `quality_5g_rssi_level`—This parameter obtains the value from the `repacd.WiFiLink.independent_channel_selection_rssi_level` UCI command.
- `rssi_check_total_counter`—This parameter obtains the value from the `repacd.WiFiLink.independent_channel_selection_total_rssi_counter` UCI command.

- `is_24g_backhaul_down`—This parameter indicates whether the 2.4 GHz backhaul is down or up. 0 denotes that 2.4 GHz backhaul is down, whereas 1 denotes that 2.4 GHz backhaul is up.
- `is_5g_backhaul_down`—This parameter indicates whether the 5 GHz backhaul is down or up. 0 denotes that 5 GHz backhaul is down, whereas 1 denotes that 5 GHz backhaul is up.
- `rssi_counter`—This parameter indicates the number of times RSSI quality check is performed. A value of 0 denotes 0xFF, which disables the 5G backhaul RSSI quality check.
- `rssi_level_debug`—This parameter obtains the value from the repacd.WiFiLink.independent\_channel\_selection\_rssi\_debug UCI command. A value of 0 denotes 0x01, which displays the RSSI level.

The following modifications are made to the repacd script:

- If repacd status shows the state is in 'RE\_LocationSuitable', the verification check as outlined in the following steps is performed. If the state is changed, all the statuses are reset.
- If repacd calls the interface down for 2.4G backhaul by using `_repacd_wifimon_bring_iface_down`, then set the in-network information into driver and start ACS for 2.4G front haul. Set flag `is_24g_backhaul_down` to 1 and set `rssi_counter` to 0.
- If repacd calls the interface down for 5G backhaul by using `_repacd_wifimon_bring_iface_down`, if `is_24g_backhaul_down` is true, and if `force_down_24g` is 0, then 2.4G backhaul is brought up. Set flag `is_5g_backhaul_down` to 1 and set `rssi_counter` to 0xFF.
- If `rssi_counter` is not 0xFF, if `is_24g_backhaul_down` is 0, `is_5g_backhaul_down` is 0, `force_down_24g` is 0, and `force_down_5g` is 0, start the RSSI quality check. If 5G backhaul is stable, then call `_repacd_wifimon_bring_iface_down` to bring down the 2.4G backhaul.
- If `rssi_counter` is not 0xFF, `is_24g_backhaul_down` is 1, `is_5g_backhaul_down` is 0, `force_down_24g` is 0, and `force_down_5g` is 0, start the RSSI quality check. If 5G backhaul is reduced, then call `_repacd_wifimon_bring_iface_up` to bring up the 2.4G backhaul and set `rssi_counter` to 0.

### 19.12.1 Channel planning on 2.4 GHz bands

The following are the sample configuration scenarios of channel planning on 2.4 GHz band:

- Case 1 - When satellite is powered up or backhaul is switched back to 5 GHz

Satellites connect to the base station or other satellites using the usual mechanism for best uplink node selection. After the network has stabilized and backhaul has been selected, 5 GHz backhaul is determined to have RSSI better than the threshold. Satellite should provide topology information to driver for input to ACS algorithm (ieee\_acs.c). A mechanism needs to be provided to access this information. 2.4 GHz STA interface on Satellite should be brought down. Channel selection is triggered on 2.4 GHz. Customers are responsible for coming up with modified algorithm, which takes topology information into consideration. The channel is changed to the new 2.4 GHz channel immediately or when promoted to do so by an application.

- Case 2: When satellite backhaul is 2.4 GHz

Do not perform channel selection as the 2.4 GHz radio is the fronthaul and the backhaul. Periodically, backhaul logic selection for 5 GHz is initiated.

- Case 3: When 5 GHz backhaul goes down

If the 5 GHz backhaul link goes down for some reason, satellite must detect it and bring up the 2.4 GHz STA interface. STA interface is expected to scan and connect to best uplink node.

Channel of the 2.4 GHz radio needs to change to the 2.4 GHz channel of the upstream node it connects to. If 5 GHz backhaul goes down, wait for X duration. If both 2.4GHz and 5GHz are up, disconnect STA on 2.4 GHz. Take a timeout before rescanning for new fronthaul channel for RE. This delay should not come into play on the boot-up case.

### 19.12.2 Channel planning on 5 GHz bands

In a tri-radio AP/RE, where one 5GHz is the backhaul and a separate one is used for the fronthaul, executing channel planning can cause the front haul radios to be on different channels in a distributed Wi-Fi network. The requirement is that seamless roaming should work in these conditions.

To support roaming between different APs on different channels:

- For each STA, periodically calibrate the relationship between downlink and uplink RSSI:
  - Perform a few 11k measurements and determine the approximate relationship between uplink and downlink RSSI values.
  - Make sure that the uplink RSSI value is recent when performing this calibration.
- When AP steering threshold is crossed:
  - Convert the serving uplink RSSI value that triggered the steering to a downlink absolute (in dBm) value using the conversion factor for that STA.
  - Perform 11k measurement on a non-serving channel. This may need to round robin between different non-serving channels if there is more than one non-serving channel in use.
  - Proceed normally when determining whether the min RSSI increase is met.

### 19.12.3 Sample configuration scenarios

This section describes the following sample network deployments:

#### Disable the 2.4 GHz backhaul if 5 GHz backhaul quality is good

Consider a sample network topology in which RE disables the 2.4 GHz backhaul if 5 GHz backhaul quality is good. The prerequisite in this network deployment is that RE must have only one 5 GHz backhaul connection and 2.4 GHz fronthaul connection run for ACS. Configure APUT1 as CAP and APUT2 as RE in Wi-Fi SON mode using auto-configuration. Enter the following UCI commands:

```
uci set repacd.WiFiLink.independent_channel_selection_enable=1
uci set repacd.WiFiLink.independent_channel_selection_rssi_level = -70
uci set repacd.WiFiLink.independent_channel_selection_total_rssi_counter
= 10
uci set repacd.WiFiLink.independent_channel_selection_rssi_debug =1
uci commit
```

The configuration is successful.

Connect both APs using WPS push button. The connection is successful.

Verify that hyd and wsplcd are running on both CAP and RE by entering the following commands:

- hyd is running with “ps | grep hyd”
- wsplcd is running with “ps | grep wsplcd”

On RE side, use “iwconfig athx” command where x is for 5G backhaul interface to make sure 5G signal quality is higher than -70. In console log, the rssi\_counter value is recorded. If rssi\_counter value is higher than 10, RE must run the ACS. The ACS log can be determined on the console. In such a scenario, 2.4G backhaul is down and only 5G backhaul is working.

On RE side, use wifitool athx get\_in\_network\_info command where x is 2.4G fronthaul interface to check whether the in-network information is set in driver. The MAC address and channel information can be determined as “XX:XX:XX:XX:XX:XX CH”.

### **Enable the 2.4 GHz backhaul if 5 GHz backhaul quality is degraded**

Consider a sample network topology in which RE enables the 2.4 GHz backhaul if 5 GHz backhaul quality is degraded. The prerequisite in this network deployment is that RE 2.4 GHz backhaul must be up if 5G backhaul quality becomes poor. Configure APUT1 as CAP and APUT2 as RE in Wi-Fi SON mode using auto-configuration. Enter the following UCI commands:

```
uci set repacd.WiFiLink.independent_channel_selection_enable=1
uci set repacd.WiFiLink.independent_channel_selection_rssi_level = -70
uci set repacd.WiFiLink.independent_channel_seleciton_total_rssi_counter
= 10
uci set repacd.WiFiLink.independent_channel_seleciton_rssi_debug =1
uci commit
```

The configuration is successful.

Connect both APs using WPS push button. The connection is successful.

Verify that hyd and wsplcd are running on both CAP and RE by entering the following commands:

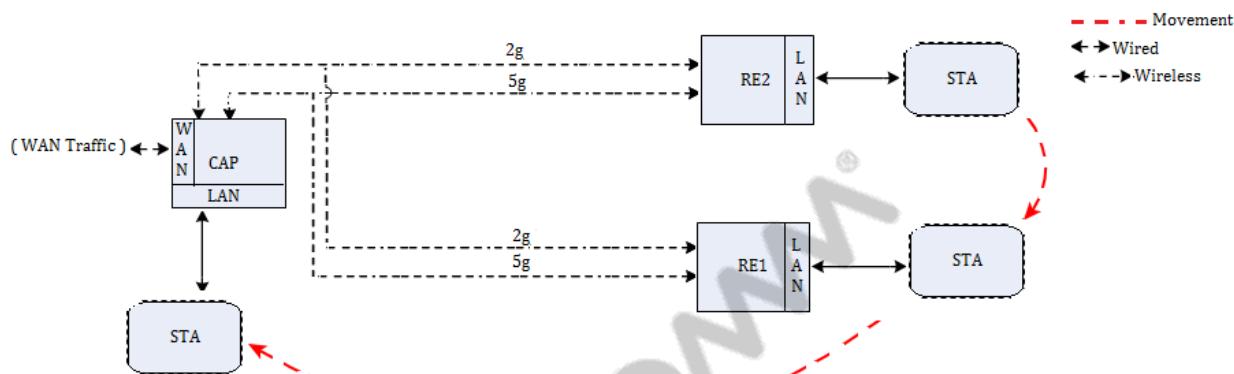
- hyd is running with “ps | grep hyd”
- wsplcd is running with “ps | grep wsplcd”

On the RE side, use command “iwconfig athx” where x is for 5G backhaul interface to make sure 5G signal quality is higher than -70. In console log, the rssi\_counter value is recorded. If rssi\_counter value is higher than 10, RE must run the ACS. The ACS log can be determined from the console. In such a scenario, 2.4 GHz backhaul is down and only 5 GHz backhaul is working.

Move RE farther from the CAP and use “iwconfig athx” command where x is for 5G backhaul interface to make sure 5G signal quality is less than 30. In console log, the rssi\_counter value is recorded. If rssi\_counter value is higher than 10, both 2.4 GHz backhaul and fronthaul must use the same channel. STA is able to connect to the 2.4 GHz fronthaul.

## 19.13 Wi-Fi SON: Ethernet roaming per port

In a Wi-Fi SON network, Ethernet roaming per port functionality is supported.



**CAP** can connect with more than one **RE**. Topology Discovery sends IEE1905 discovery messages to each **RE** and **CAP** to populate the topology discovery database for next-hop reachability using all the available interfaces.

Wired STA/PLC connection can be unplugged at any point and the unplugged STA/PLC can join any other **RE** or **CAP** directly in SON network. **CAP/RE** bridge is expected to aware about MAC moving from one interface to another, such as moving behind br-lan or a single grouping port.

If a PLC network, the link up/down event does not work because the link is always up and the station moves behind the PLC network.

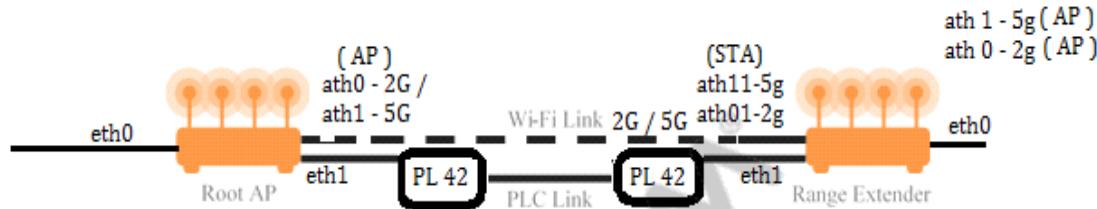
The following VID changes are required on IPQ8064 devices (AP161, AP148), which depends on the /etc/network config of IPQ8064 platforms:

```
uci set hyd.@Vlanid[0].vid='2'
uci set hyd.@Vlanid[1].vid='1'
uci commit hyd
do system reboot
```

## 19.14 Wi-Fi SON: PLC backhaul

| Feature                            | IPQ4019.ILQ.1.2.2 |
|------------------------------------|-------------------|
|                                    | IPQ4019           |
|                                    | 3.14              |
| Wi-Fi SON support with PLC Phase I | ✓                 |

The PLC backhaul provides automatic path selection between Wi-Fi interfaces and PLC link. The PLC link is the third interface between RootAP and Range Extender in addition to Wi-Fi 2 GHz and Wi-Fi 5 GHz.



RootAP does the CAP functionality where eth0 is the WAN gateway.

The following interfaces in the br-lan network share the same local IP address. PLC is connected to any one of following LAN ports:

- eth1 interface is for PLC
- ath0 interface is for 2 GHz
- ath1 interface is for 5 GHz

Range Extender does the RE role where eth0 and eth1/PLC, and AP (AP VAPs are ath1 for 5 GHz, ath0 for 2 GHz) are connected to bridge.

- eth0 interface for Ethernet
- eth1 interface is for PLC
- ath01 interface is for 2 GHz -STA Vap
- ath11 interface is for 5 GHz - STA Vap
- ath1 interface is fro 5 GHz - Ap VAP
- ath0 interface is for 2 GHz - Ap VAP

The following are the IPKs available:

- qca-plc-fw\_00015\_ipq806x.ipk
- qca-plc-fw-7500\_00015\_ipq806x.ipk
- qca-plc-serv\_g2a5cf60-1\_ipq806x.ipk

### Installation procedure

1. These ipks are loadable modules. Copy the ipks into Host/Target platform using the following commands:

```
opkg install qca-plc-fw_00015_ipq806x.ipk
opkg install qca-plc-fw-7500_00015_ipq806x.ipk
opkg install qca-plc-serv_g2a5cf60-1_ipq806x.ipk
```

2. After the installation is successful, enable PLC host software using the OpenWrt UCI framework using the following commands:

```
uci set plc.config.Enabled='1'
uci set plc.config.AggrLinkRate='50' '0' /*50 0 is the default value to
support dynamic plc link metric */
/* standalone PLC dongle does not require PLC firmware.
```

PIB is flushed already to form the PLC Link between Root AP and Range Extender. \*/  
`uci set plc.config.FwPib_Download='0'`

For REH172 platforms, enter the following:

```
uci set plc.config.FwPib_Download='1'
```

3. Start PLC using the following command:

```
/etc/init.d/plc enable
/etc/init.d/plc start
```

To stop PLC, use the following command:

```
/etc/init.d/plc stop
```

To restart PLC, use the following command:

```
/etc/init.d/plc restart
```

To upgrade the PLC link capacity rate, set ‘AggrLinkRate’ option with the appropriate aggregated Link Capacity for the PLC network. The third-party software should determine the appropriate value of Link Capacity for the given PLC network. Set the appropriate value for AggrLinkRate option by using the UCI command in step 2. Restart PLC to ensure the new value is taken for the Automatic Path selection.

Topology discovery messages are exchanged between RootAP and Range Extender to detect the available interfaces which are directly connected. After successful detection of the three links (Wi-Fi 2 GHz, Wi-Fi 5 GHz and PLC), the traffic switches over to the best interface.

Before starting Wi-Fi SON configuration, do the following:

Set PLC interface name on CAP and RE side using the following commands:

```
uci set plc.config.PlcIfname='eth1'
uci commit
```

For AP152 + QCA9886 platforms, set PLC interface name on CAP and RE side using the following commands:

```
uci set plc.config.PlcIfname='eth0.1'
uci commit
```

In RE, attach eth0 to br-lan by using following command:

```
uci set network.lan.ifname ='eth1 eth0'
uci commit
```

For AP152 + QCA9886 platforms, in RE, attach eth0 to br-lan by using following command:

```
uci set network.lan.ifname ='eth1 eth0.2'
uci commit
```

When only PLC is used as backhaul, use the following command on RE for autoconfiguration cloning using the PLC interface.

```
uci set repacd.repacd.DefaultREMode='son'
uci commit
```

Disable QRFS using the following commands:

```
/etc/init.d/qrfss stop
/etc/init.d/qrfss disable
```

For REH172 platform The following PIBs were included in PLC ipks for QCA7550/QCA7520 support:

- QCA7550-REH172\_HomePlugAV\_NorthAmerica.pib
- QCA7520-REH172\_HomePlugAV\_NorthAmerica.pib
- QCA7550-REH172\_EN50561-1.pib
- QCA7520-REH172\_EN50561-1.pib

To change PIB, enter the following commands:

```
uci set plc.config.PibPath='/etc/plc/<QCA7500-REH172_HomePlugAV_
NorthAmerica>.pib'
uci commit
```

Remove all the files under /etc/plc/ directory and reboot the system.

**NOTE** For more details on other Wi-Fi SON commands, refer to *AP 10.4 Command Line Interface (CLI) User Guide* (80-Y8052-1).

## ESS port mapping in REH172

On REH172, the machine ID of the device must be set as 8010007. By default, REH172 runs in non-SON mode. The following table describes the mapping of physical port numbers with the Linux interface names for REH172 in non-SON Mode. Physical port mappings are controlled by ESS.

| ESS port mappings in REH172 (DK05 + PLC)   |                   |                           |
|--|-------------------|---------------------------|
| Non-SON mode   |                   |                           |
| Interface  | Connectivity      | ESS/Physical port numbers |
| Eth0<br>On REH172 platforms for non-SON mode, only one Linux interface, eth0, is shared for both physical ports. | PLC               | 0, 4                      |
| Eth0   | External LAN Port | 0,5                       |

The following table describes the mapping of physical port numbers with the Linux interface names for REH172 in SON Mode. Physical port mappings are controlled by ESS.

| ESS port mappings in REH172 (DK05 + PLC) |              |                  |
|--|--------------|------------------|
| SON mode                                 |              |                  |
| Interface                                | Connectivity | Ports connection |
| Eth1                                     | LAN          | 0,1,2,3,4        |
| Eth0                                     | WAN          | 0,5              |

- On REH172, use the following procedure to configure the **Wi-Fi SON** mode; the commands can be added in startup script in the platform.

```
cd /proc/sys/net/edma/
echo 0x20 > default_group1_bmp
echo 0x1E > default_group2_bmp
echo 2 > default_group1_vlan_tag
echo 1 > default_group2_vlan_tag
cd /* go back to root dir,*/
```

- Edit /etc/config/network and ensure that the ports are in the right group.

```
<snip>
config switch
    option name 'switch0'
    option reset '1'
    option enable_vlan '1'
config switch_vlan
    option device 'switch0'
    option vlan '1'
    option ports '0t 1 2 3 4'
config switch_vlan
    option device 'switch0'
    option vlan '2'
    option ports '0t 5'
<snip>
```

- eth0 can be added in the bridge by using brctl addif br-lan eth0, or by entering the uci set network.lan.ifname='eth1 eth0' command.
  - Enter the uci commit command to save the configuration.
  - Reboot the system after it is verified.

- On REH172, use the following procedure to configure the **non-Wi-Fi SON** mode; the commands can be added in startup script in the platform. This configuration is already by default on REH172.

```
cd /proc/sys/net/edma/
echo 0x3e > default_group1_bmp
```

```

echo 0x0 > default_group2_bmp
echo 0 > default_group1_vlan_tag
echo 0 > default_group2_vlan_tag
cd /* go back to root dir,*/
 Edit /etc/config/network and ensure that the ports are in the right group.
config switch
    option name 'switch0'
    option reset '1'
    option enable_vlan '0'
config switch_vlan
    option device 'switch0'
    option vlan '0'
    option ports '0 1 2 3 4 5'
#config switch_vlan
#option device 'switch0'
#option vlan '2'
#option ports '0t 5'
 Reboot the system after it is verified.

```

### 19.14.1 PLC configuration

- To enable PLC on REH172, you need to make sure that the following ipk have been installed:

```

opkg install qca-plc-fw_00046_ipq806x.ipk
opkg install qca-plc-fw-7500_00046_ipq806x.ipk
opkg install qca-plc-serv_g2a5cf60-1_ipq806x.ipk

```

- Then, you need to use the following commands:

```

uci set plc.config.Enabled='1'
uci set plc.config.FwPib_Download='1'
uci set plc.config.PibPath='/lib/firmware/plc/7500/QCA7500-REH172_
EN50561-1.pib'
uci commit plc

```

### PIB information

In the REH172 platform, following PIBs are included for QCA7550/QCA7500 support in qca-plc-fw-7500\_00046\_ipq806x.ipk (under /lib/firmware/plc/7500/)

- QCA7550-REH172\_HomePlugAV\_NorthAmerica.pib
- QCA7500-REH172\_HomePlugAV\_NorthAmerica.pib
- QCA7550-REH172\_EN50561-1.pib
- QCA7500-REH172\_EN50561-1.pib

To change the PIB:

1. Run the following commands:

```
uci set plc.config.PibPath='/lib/firmware/plc/7500/QCA7550-REH172_
EN50561-1.pib'
uci commit
```

1. Remove all the files under **/etc/plc/** directory.
2. R
3. eboot the system.

### 19.14.2 PLC SON interface mapping

- Before doing the PLC SON configuration, set PLC interface name on CAP and RE side:

```
uci set plc.config.PlcIfname='eth1'
uci commit
```

eth1 is used for PLC Interface for PLC backhaul connectivity between CAP and RE.

- In the repeater, PLC and one Ethernet port can be added in the bridge networks along with STA, AP VAPs:

```
uci set network.lan.ifname ='eth1 eth0'
uci commit
```

- When only PLC is used as backhaul between CAP and RE, use the following command on RE for auto configuration cloning thru PLC interface:

```
uci set repacd.repacd.DefaultREMode='son'
uci commit
```

**NOTE** APS switching from WLAN to PLC does not work with the existing overloaded single-stream traffic flow. This issue not seen if multiple-stream traffic is used.

**NOTE** Low throughput observed with SON. SON runs on user space level with all features of SON. A 15% impact in throughput is observed.

### 19.14.3 Fronthaul AP VAPs and CAP reachability

When CAP is not reachable, the fronthaul AP VAPs of RE will be brought down after 5 mins (configurable parameter in repacd.config file). If CAP is reachable, the Fronthaul AV VAPs of RE will be brought up immediately. The FrontHaulMgr section is added in /etc/config/repacd to control Fronthaul AP VAPs based on CAP's reachability.

```
repacd.FrontHaulMgr.ManageFrontAndBackHaulsIndependently='1'
repacd.FrontHaulMgr.FrontHaulMgrTimeout='300'
```

- `ManageFrontAndBackHaulsIndependently` is used to enable or disable the feature to control RE's Fronthaul AP VAPs based on CAP's reachability.
- `FrontHaulMgrTimeout` is expiry time to bring down REs Fronthaul AP VAPs if CAP is not reachable

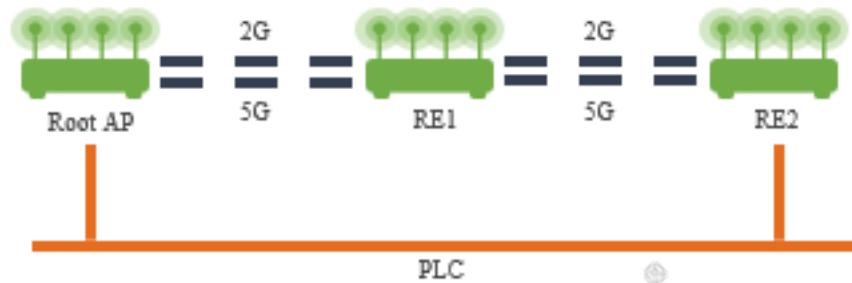
#### 19.14.4 Guidelines for configuring Wi-Fi SON interoperation with PLC

- PLC SON and Ethernet backhaul/loop prevention cannot coexist. PLC SON and Ethernet backhaul are mutually exclusive.
- VLAN ID configuration for PLC interface is not supported in PLC SON.
- AVitar/EDM tool is not supported in PLC SON.
- Guest and private networks are not supported in PLC SON.
- When different PLC chipsets are used in the Wi-Fi SON network (for example, CAP is connected to QCA7500 and RE is connected to AR7420), the behavior will not be as expected.
- In daisy chain PLC SON, PLC link-failover (whenever PLC links are present on CAP, RE1, and RE2) at CAP side, and load-balancing between WLAN and PLC are not supported.
- If QCA9886 AP152 + QCA7500/QCA7550/QCA7420 platform runs with 90-100% CPU utilization in SON network, there might be topology discovery entry of PLC changing frequently due to CPU busy handling traffic. The topology discovery packet is a keepalive message between CAP and repeater.
- RFC2544 throughput impact is observed for lower-size packets (64 bytes, 128 bytes) with EN50561 PIB. This reduced performance for smaller packet size is due to a CPU busy detection feature implemented in boosted PSD firmware, to address the dips in UDP SNR. The fix involved adding a new sequence to invoke the CPU busy detector since the tone map generation tasks did not get the CPU time to run. This is a system limitation.

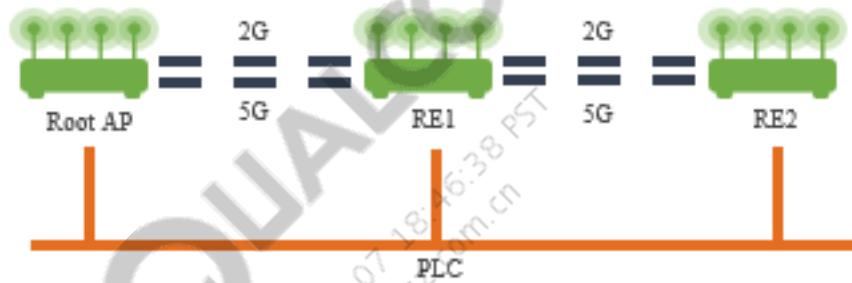
### 19.15 Wi-Fi SON with PLC: Support for daisy-chain and star topologies

Enhancements are introduced for the Wi-Fi SON integration with PLC capability. Support for daisy chain topology is implemented for interoperation of Wi-Fi SON with PLC, which allows an RE to connect to another RE and to extend the range further in Wi-Fi links in conjunction with PLC interfaces. Also, support for star topology is introduced, which enables a CAP to be connected to all repeaters or REs using Wi-Fi and PLC interfaces.

The following diagrams illustrate a simple daisy chain network, where RE1 is connected to the CAP and RE2 is connected to RE1 along with PLC interfaces. Note, both REs have HYD instances running, and access point (AP) steering and band steering take place as these operations normally occur.

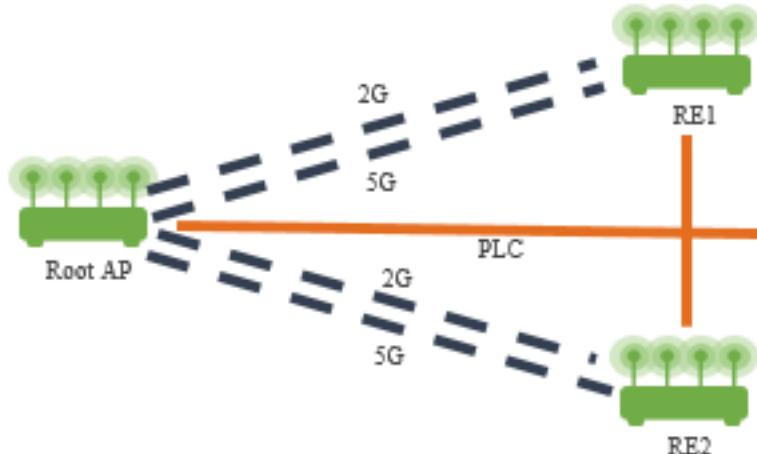


**Figure 19-7** Daisy chain support for interconnection of CAP and RE2 using Wi-Fi and PLC, and RE1 using Wi-Fi only



**Figure 19-8** Daisy chain support for interconnection of CAP, RE1, and RE2 using PLC and Wi-Fi

The following figure illustrates shows a star network, where RE1 and RE2 are connected to the CAP along with PLC interfaces. Note, both REs have HYD instances running, and AP steering and band steering processes take place as these operations normally occur.



**Figure 19-9** Star topology for interconnection of REs and CAP with Wi-Fi SON and PLC

### 19.15.1 Topology building process

Topology message exchange and processing makes the core of daisy chain support. 1905 Topology messages are used to learn about network topology by all nodes in the network. The following are topology messages:

1. Topology Discovery: Neighbor-Multicast
  - a. Sent by each node on all interfaces periodically
  - b. Consumed by direct neighbor, never forwarded
  - c. Helps build Direct Neighbors
2. Topology Notification: Multicast-Relay
  - a. Sent whenever a node detects change on its topology or interface i.e. new link up/down, STA associated/disconnected etc.
  - b. Relayed by all nodes, nodes discard previously seen Notifications to prevent loop
3. Topology Query: Unicast
  - a. Sent by a node to a specific destination to query of its current state and interfaces
  - b. Periodically sent for each entry in Database
4. Topology Response: Unicast
  - a. Sent in response to Topology Query to querying node
  - b. Provides its current state
  - c. Used by both Direct and Distant Neighbors
  - d. Has following TLVs
    - i. Device Info
    - ii. Capabilities
    - iii. List of Legacy Neighbors (for each of its interfaces)
    - iv. 1905 Device Neighbors on its interfaces
    - v. Attached WLAN Stations

The topology discovery/query/response is exchanged over Wi-Fi and PLC interfaces. All PLC nodes are in the same HomePlug AV Logical Network (AVLN).

### 19.15.2 Supported configuration scenarios

This section describes the supported configuration topologies in Wi-Fi SON integration with PLC with daisy-chain topologies that contain REs connected to root AP or CAP

#### **Scenario 1: Daisy chain support for interconnection of CAP and RE2 using Wi-Fi and PLC, and RE1 using Wi-Fi only**

- RE1 is configured as DK01/DK07.
- CAP connected directly to RE1 through Wi-Fi, RE2 through PLC

- RE1 connected directly to CAP through Wi-Fi, RE2 through Wi-Fi
- RE1 upstream device is CAP, all upstream traffic goes through Wi-Fi
- RE2 connected directly to RE1 through Wi-Fi, CAP through PLC
- RE2 upstream device is RE1, all upstream traffic goes through Wi-Fi
- The Wi-Fi link between the nodes uses both bands 2G and 5G.
- RE2 PLC link is down all the traffic should switchover to Wi-Fi. After the PLC link is up, adaptive path selection (APS) chooses PLC link for new stream, and the existing stream can switchover after oversubscribed in Wi-Fi link.

**Scenario 2: Star topology for interconnection of REs and CAP with Wi-Fi SON and PLC**

- CAP connected directly to RE1 through Wi-Fi and PLC
- RE1 connected directly to CAP through Wi-Fi and PLC
- RE1 upstream device is CAP
- RE2 connected directly to CAP through Wi-Fi and PLC
- RE2 upstream device is CAP
- The Wi-Fi link between the nodes uses both bands 2G and 5G.
- RE1 is directly connected to RE2 through PLC.
- Repeaters should not see other repeater as upstream device.

**Scenario 3: Daisy chain support for interconnection of CAP, RE1, and RE2 using PLC and Wi-Fi**

- REH172 is used as RE1.
- CAP connected directly to RE1 through Wi-Fi and PLC
- CAP connected directly to RE2 through PLC
- RE1 connected directly to CAP through Wi-Fi and PLC
- RE1 connected directly to RE1 through Wi-Fi and PLC
- RE1 upstream device is CAP
- RE2 connected directly to RE1 through Wi-Fi and PLC
- RE2 upstream device is RE1
- The Wi-Fi link between the nodes uses both bands 2G and 5G.

### 19.15.3 PLC Link Metrics

Aggregate link capacity rate and medium utilization for PLC interface are supported. Dynamic Link Metrics is supported for PLC devices.

The following usage guidelines apply for full support of Dynamic Link Metrics for PLC devices:

1. PLC interface MAC address must be the same as PLC device MAC address.
  - a. For an external PLC device, the MAC address of PLC device learned at Initialization of PLCHOSTSRV is updated as local MAC address. (The OUI of PLC device is addressed.)
  - b. For REH172, the host's br-lan MAC address + 2 is applied to PLC chipset.
2. Chipset ID is obtained from the PLC device through management message entry (MME) and the threshold value is set for Medium Utilization.
3. Cumulative Phy Rate Tx/Rx is required for more than 2 PLC nodes in the AVLN.
4. Dynamic Link metrics is expected to work for PLC device of same chipset ID.
5. A combination of AR742x and QCA75xx chipsets in the same AVLN causes data imbalance, while calculating the Average Link Capacity.
6. Perform the following workaround to test the Dynamic Link Metrics in PLC
  - a. Set the PLC interface MAC address to PLC device using EDM
  - b. Set the fixed MAC address in IPQ4019 either using UCI commands or at the uboot prompt.

#### **19.15.4 Band/Interface selection for AP steering**

AP steering is dependent on path selection, band selection and the position of CAP/RE1/RE2 from Client that needs to roam between CAP/RE1/RE2 in the forward and backward directions. If path selection chosen is Wi-Fi, a slight delay might occur in packets due to hop-to-hop switchover.

- Dual band usage between CAP -- RE1, and RE1 -- RE2
- Single band usage either 2G/5G across the network between CAP -- RE1, and RE1 -- RE2
- The scenario with a combination of single band usage between CAP -- RE1, and RE1 -- RE2 in 2 GHz and 5 GHz bands is not currently supported.
- In an environment with a mixture of single band usage between CAP -- RE1, and RE1 -- RE2 in 2 GHz and 5 GHz bands, disable intra-BSS bridging or AP bridging to verify whether the setting works.
- Mixed interface usage between CAP -- RE1 (Wi-Fi link), and CAP -- RE2 (PLC link) is not currently supported.

#### **19.15.5 Configuration commands to enable Dynamic Link Metrics**

No code changes are made in PLC firmware or PLC Host Server daemon to support daisy-chain and star topologies with Aggregate Link Rate Capacity. To enable Dynamic Link Metrics, set the following parameters before starting hyd and PLC daemon:

- In /etc/config/hyd, the HostPLCInterfaceSpeed should be set to
  - 100 for AR742x Ethernet speed 100Mbps (uci set hyd.PathChPlc.HostPLCInterfaceSpeed='100')

- 500 for AR742x Ethernet Host Speed 1Gbps (uci set hyd.PathChPlc.HostPLCInterfaceSpeed='500')
- 1000 for QCA75xx Ethernet Host Speed 1Gbps (uci set hyd.PathChPlc.HostPLCInterfaceSpeed='1000')
- In /etc/config/hyd , the MaxMediumUtilization should be set to
  - 40 for AR742x Ethernet speed 100Mbps (uci set hyd.PathChPlc.MaxMediumUtilization='40')
  - 80 for AR742x Ethernet Host Speed 1Gbps (uci set hyd.PathChPlc.MaxMediumUtilization='80')
  - 80 for QCA75xx Ethernet Host Speed 1Gbps (uci set hyd.PathChPlc.MaxMediumUtilization='80')
- In /etc/config/plc AggrLinkRate should be set to zero.
  - uci set plc.config.AggrLinkRate='0'

## 19.15.6 Guidelines and limitations

When Wi-Fi SON with PLC is enabled on REH172 or on IPQ4019 + standalone PLC, the following guidelines and limitations apply:

- With DK01+PL42, no problems are observed with topology and ping between the nodes (CAP/RE1/RE2). The path selection for traffic flow through PLC between the nodes is based on the Aggregate Link Rate configured. The Dynamic Link Metrics with AR742x and QCA75xx is verified to work properly with 100 Mbps as interface speed on AR742x, 1G as host interface speed on QCA75xx, and REH172 machine ID set to DK01.
- Restart host when swapping one PLC device to another PLC device.
- If standalone PLC is disconnected at CAP, then at RE the PLC entry is removed in td table. At this time, the ping packets or traffic will not flow in PLC link to other devices connected through PLC devices.
- The firmware and parameter information block (PIB) are not downloaded if the eth1 broadcast (bcast) bit is set to disabled.
- If PLC device asserts, topology discovery does not occur during that time. At the repeater side, the eth1 bcast is set to disabled. During this time, the firmware and PIB are not downloaded.
- ETH-SON and PLC-SON cannot coexist. The VLAN ID configuration impact analysis results show that there is no improvement in the existing behavior. Support for VLAN ID for PLC interface is not implemented.
- PLC roaming is not supported (that is, disconnection of one PLC device and connection of another PLC device is not supported).
- Using the td s2 topology display, NID value is shown as 00. Currently, the NID value is not used by PLC host in SON network. This value can be disregarded.

## 19.16 Wi-Fi SON: Determine the join and leave events of peers

In a Wi-Fi SON topology, a mechanism to identify the time periods at which a peer joins or leaves the network is implemented. Enhancements are made to introduce new events in the Wi-Fi SON module and to add a module to the Hy-Fi daemon (HYD) to send events to a customer daemon when the customer registers the event. The customer can either use an existing daemon or create a new one to receive notifications regarding the joining and leaving of peers in the Wi-Fi SON network from HYD customer module.

### 19.16.1 Addition of events to Wi-Fi SON module

The following two files are added to qca/src/qca-wifison-ext-lib

- wifison-event.c
  - The following four-function body is supported:
    - wifison\_event\_init—Reference the plchost to create a link with hyd.
    - wifison\_event\_deinit—Destroy the link with hyd.
    - wifison\_event\_register—if event is register, the hyd customer module will trigger this event notification
    - wifison\_event\_deregister—if event is deregister, the hyd customer module will not trigger this event notification.
    - wifison\_event\_get—Types of events that hyd customer module sends.
- wifison-event.h
  - The following function prototype is supported:
    - wifison\_event\_init
    - wifison\_event\_deinit
    - wifison\_event\_register
    - wifison\_event\_deregister
    - wifison\_event\_get

Other define directives are included here because the customer needs to include this header file.

If the customer daemon needs to use those events, it is necessary to link to libhyficommon.so and include the header file to call those four functions.

### 19.16.2 Addition of sonevent module into HYD

HYD contains several modules and each module had its own job and each module is using event to do communication. For meeting the HYD modularization design, one more module called sonevent module is introduced. This module will focus to send event to the customer daemon if customer registers the event.

When TD module found the RE join/leave, it will send an event to sonevent module. The sonevent module determines whether the event is registered or not. If the event is registered, a notification is sent to the customer daemon. If it is necessary to add more events (such as steering successful/failed), other modules can send an event to sonevent module and sonevent module can handle it to notify the customer daemon.

### 19.16.3 Process of HYD communication with customer daemon

The following processes occur in a sequence when HYD module communicates with the customer daemon:

1. When customer daemon call the wifison\_event\_init, it will create a socket.
2. When HYD init the sonevent module, it will also create a socket and also create a buffer read thread (which is the same as plcManagerInit).
3. When customer daemon calls wifison\_event\_register, it will use send/sendto functions in the code to the HYD sonevent socket. The sonevent module needs to recode the event to be registered.
4. When HYD other modules send a notification event to the sonevent module, sonevent module parses it and check is the event is register or not. If registered, it will use send/sendto functions to the socket that are created by the wifison\_event\_init.
5. Customer daemon will call wifison\_event\_get. This function performs the socket-receipt check from the HYD sonevent module. When data is received from HYD sonevent module, this module parses it and returns the event ID to customer daemon. The customer daemon implements the necessary action based on event ID.

### 19.16.4 Use the customer library in the HYD customer module

If a customer already has daemon running in background, the customer can create a new thread to get those notifications from HYD customer module. The following is a sample code snippet:

```

pthread_t event_notification;
pthread_create(&event_notification, NULL, event_notification, data);
pthread_join(event_notification, NULL);
void * event_notificaiton (void *data)
{
    int socket;
    int wifison_event_id = -1;
    socket = wifison_event_init();
        wifison_event_register (event_id_0);
        wifison_event_register (event_id_1);
        while (1) {
    wifison_event_id = wifison_event_get(socket);
    if (wifison_event_id== -1)
    {
        printf("event get failed\r\n");
        goto err;
    }
        switch (wifison_event_id) {
            case even_id_0:
                break;

```

```

        case even_id_1:
            break;
    }
err:
    wifison_event_deinit();
    wifison_event_deregister(event_id_0);
    wifison_event_deregister(event_id_1);

}

```

If customer does not have any daemon, create a new daemon to run in the background. The following is a sample code snippet:

```

int main(int argc, char **argv)
{
    int socket;
    int wifison_event_id = -1;
    socket = wifison_event_init();
        wifison_event_register (event_id_0);
        wifison_event_register (event_id_1);
        while (1) {
    wifison_event_id = wifison_event_get(socket);
    if (wifison_event_id== -1)
    {
        printf("event get failed\r\n");
        goto err;
    }
        switch (wifison_event_id) {
            case even_id_0:
                break;
            case even_id_1:
                break;
        }
    }
err:
    wifison_event_deinit();
    wifison_event_deregister(event_id_0);
    wifison_event_deregister(event_id_1);

}

```

The sample daemon code is added into qca/src/qca-wifison-ext-lib/sample/wifison\_event\_sample.c. Customers need to use these sample code blocks to add it into their daemon to get events. Support is not currently available to assist customers to modify their makefile and integrate them; users must perform these changes.

### 19.16.5 Verify the join and leave events of one RE

Consider a sample topology in which CAP and RE\_0 are up and running. CAP can determine the event when RE\_0 joins the network. To enable CAP determine the joining of RE\_0, do the following:

1. Configure APUT1 as CAP and APUT2 as RE0 in SON mode using auto-configuration. When CAP is up, run “wifisonevent &” in background.  
The configuration is successful.
2. Connect both APs using WPS push button.  
The connection is successful.
3. Verify that hyd is running on both CAP and RE using the “ps | grep hyd” command, and that wsplcd is running using the “ps | grep wsplcd” command.  
The output of these commands verify that hyd and wsplcd are running.
4. The console displays one line as “RE XX:XX:XX:XX:XX join” where XX:XX:XX:XX:XX is for RE bridge MAC address.  
The console displays the join message.
5. Manually bring down the RE 5G/2.4G backhaul interface by “wpa\_cli -p /var/run/wpa\_supplicant-athX disable\_network 0” where X is for 5G/2.4G backhaul interface.  
The console displays one line as “RE XX:XX:XX:XX:XX leave”, where XX:XX:XX:XX:XX is for RE bridge MAC address. The console displays the RE leave message.

### 19.16.6 Verify the join and leave events of two REs

Consider a sample scenario in which two REs are connected in a daisy-chain format (CAP <--> RE0 <--> RE1). To enable CAP determine the joining and leaving of RE0 and RE1, do the following:

1. Configure APUT1 as CAP and APUT2 as RE0/RE1 in SON mode using auto-configuration. When CAP is up, run “wifisonevent &” in background.  
The configuration is successful
2. Connect both APs using WPS push button.  
The connection is successful with daisy chain mode (CAP<-->RE0<-->RE1)
3. The console displays two lines as “RE XX:XX:XX:XX:XX join” where XX:XX:XX:XX:XX is for RE0/RE1 bridge MAC address.  
This join message on the console verifies that two REs are now connected to the Wi-Fi network.
4. Manually bring down the RE1 5G/2.4G backhaul interface by entering “wpa\_cli -p /var/run/wpa\_supplicant-athX disable\_network 0” command, where X is for 5G/2.4G backhaul interface. The console displays one line as “RE XX:XX:XX:XX:XX leave” where XX:XX:XX:XX:XX is for RE1 bridge MAC address.  
The console displays the leave message for one RE.
5. Manually bring down the RE0 5G/2.4G backhaul interface by entering “wpa\_cli -p /var/run/wpa\_supplicant-athX disable\_network 0” command, where X is for 5G/2.4G backhaul interface. The console displays one line as “RE XX:XX:XX:XX:XX leave” where XX:XX:XX:XX:XX is for RE0 bridge MAC address.  
The MAC address and channel information can be determined from “XX:XX:XX:XX:XX:XX CH” message on the console.

## 19.17 Wi-Fi SON: Multinode WPS support for push button events

In a Wi-Fi SON network, the capability to process the Wi-Fi protected setup (WPS) push button control (PBC) event on two Wi-Fi SON nodes is implemented. This functionality increases the possibility of a WPS client to be configured in a Wi-Fi SON network and uses the most-optimal Wi-Fi SON node to effectively process the push button events.

Before this functionality was implemented, whenever it was necessary to configure a client device in a Wi-Fi SON network with WPS, the user pressed the WPS push button on the Wi-Fi SON node that was closest to the user and the client device was configured. However, such a method did not guarantee that the best-available Wi-Fi SON node was used to establish a WPS session with the new client. In such scenarios, the WPS session fails for the client.

With this functionality introduced to process the WPS event on two Wi-Fi SON nodes, the duration of two minutes in the fronthaul VAP is divided between the root AP node and any one of the other repeater APs in the network to enhance the connection of a WPS client with the network.

### 19.17.1 Supported configuration scenarios

The following scenarios are supported with this mechanism of multinode WPS:

- When a user presses the push button on the repeater AP and the new client, the repeater fails to establish the client session in one minute, and sends an Action frame to the root. The root AP transitions to WPS state for the next one minute and if the new device is within the range of the network, the client device becomes configured.
- When a user presses the push button on the root AP and the new client, and the root AP fails to establish a session in one minute, the root AP sends an Action frame to any one of the repeater APs. This Wi-Fi SON repeater AP transitions to the WPS state for a minute and if the new device is within range, the client device becomes configured.

The hotplug scripts are the system scripts that are called as soon as a button is pushed. The order in which the hotplug scripts are run is the same as the order before the implementation of this functionality— wps-enh, wpa\_supplicant, and hostapd.

### 19.17.2 Changes in behavior

Without the multinode WPS functionality, the following behavioral guidelines applied:

1. If enhanced WPS (wps-enh) is configured, the WPS push button event is processed only by the node that received the event.
2. If enhanced WPS (wps-enh) is not configured, the WPS push button press-event is processed by the WPS Wi-Fi SON application for backhaul VAPs, by the independent hostapd for fronthaul VAPs. Also, the wpa\_supplicant is started on STA VAPs.

With the multinode WPS functionality, the following behavioral guidelines apply:

1. In a Wi-Fi SON topology:

- a. If Wi-Fi SON WPS is configured, a validation is performed to determine whether at least one STA VAP is connected. If at least one STA VAP is connected or if it is a root AP, the WPS push-button press event is sent to the Wi-Fi SON WPS application (wps-enhc script). The WPS push-button press event is not sent for processing to any other node.
  - b. If STA VAPs are present and are not connected, the enhanced WPS application (wps-enhc script) is called if it is configured. Otherwise, the wpa-suppliant application processes the call on each of the STA VAPs. The WPS push-button press event is not sent for processing to any other node.
2. In a non-Wi-Fi SON topology:
- a. The enhanced WPS application (wps-enhc script) is called. Otherwise, the hostapd and wpa\_suppliant are called independently on each VAP.

### 19.17.3 Guidelines for working with multinode WPS

Keep the following points in mind while using the multinode WPS functionality for configuring WPS clients:

1. All configuration settings are same across 2 GHz and 5 GHz bands. If one STA VAP is connected and the other STA VAP is disconnected, WPS push button event is not performed on the disconnected STA VAP.
2. This feature uses wireless medium to send action frames. Therefore, at least one wireless backhaul link must be up and operational, even if a repeater contains a PLC/ Ethernet backhaul.
3. The Wi-Fi SON WPS application is used even when one of the STA VAPs is connected. This application does not configure any of the STA VAPs. If STA VAPs must be reconfigured to a new root AP, the STA VAPs must be disconnected from the current root AP before they can be reconfigured.
4. On the repeater, when both STA VAPs are disconnected, this feature is not applicable because no method exists to inform the root AP about the event.

### 19.17.4 WPS Wi-Fi SON processing of the PBC press-event

After the Wi-Fi SON WPS application receives the WPS push-button press event, depending on whether the button is pressed on the root AP or repeater AP, the following sequence of events occur:

#### WPS PBC at root AP

1. PBC pressed at the root AP
  - a. Backhaul AP VAP—Start the Action frame-based WPS procedure by sending “WPS\_LISTEN” to all repeaters. WPS is not started on the root AP. WPS is started on BH only after repeater AP that can listen to the new client device is determined.
  - b. Front Haul AP VAP—Start the WPS on the fronthaul VAPs and also start a 1-minute timer. If WPS succeeds in the first minute, all WPA activity on the backhaul VAP is

canceled. If WPS does not complete in one minute, send an Action frame to any repeater. If the WPS starts locally in the backhaul VAP of the root, all fronthaul activity is canceled.

2. Repeater receives Action frame from root AP
  - a. BH VAP—No action on new Action frame is performed.
  - b. FH VAP—Start WPS on each of the fronthaul VAPs and start a 1-minute timer. If WPS completes within a minute, no further activity occurs. Otherwise, the timer-expiry is printed and no further activity occurs. WPS fails on both root and repeater APs.

#### **WPS PBC at repeater AP**

1. PBC pressed at the repeater AP when one or more STA VAPs are connected
  - a. Backhaul AP VAP—Start WPS on backhaul AP VAP for 1 minute. On timer expiry without WPS success, the information is sent to root AP. This method enables a nearby repeater AP to be used to configure a new client, if the chosen repeater is not the best one.
  - b. Backhaul STA VAP—No action is performed. STA is configured by other system scripts that start wpa-supplicant independently. If a user wants to reconfigure any backhaul STA, the user must disconnect both STA VAPs from the current root AP.
  - c. Fronthaul AP VAPs—Start WPS on all fronthaul AP VAPs. Run a timer for 1 minute. If WPS completes within a minute, no further activity occurs. If WPS does not complete within a minute, Action frame is sent to root AP.
2. Root AP receives Action frame from repeater AP
  - a. Backhaul VAPs—No action is performed.
  - b. Fronthaul VAPs—Start WPS on each of the fronthaul VAPs and start a 1-minute timer. If WPS completes within a minute, no further action is performed. Otherwise, the timer-expiry is printed and no further activity occurs. WPS fails on both root and repeater APs.

#### **19.17.5 Conditions when the WPS process is stopped**

The WPS process is terminated when one of the following conditions are satisfied:

1. Backhaul AP VAP of a certain remote entity receives a probe request with WPS IE set.
2. Fronthaul VAP completes the WPS process with a station.
3. Fronthaul VAP fails to complete a WPS session with a station within a minute; no activity occurs in the backhaul VAP and another entity processes for the next one minute.

#### **19.18 Wi-Fi SON: Cloning of backhaul VAP credentials for multi-SSID with traffic separation**

Support is introduced for cloning of credentials of a backhaul VAP when traffic separation is enabled. This functionality provides a way to clone backhaul VAP's credentials on repeater (RE) with the help of WPS enhancement feature when traffic separation is enabled.

Before the functionality of backhaul credential cloning for multi-SSID was implemented, on root AP, backhaul AP VAPs (part of network backhaul) received the same credentials as those of a

private VAP (part of network LAN) but with different SSIDs (with -spcl suffix). WPS was disabled on all backhaul VAPs and SSID was hidden. As a result, on RE, when traffic separation was enabled, STA VAP (part of network LAN) initially connected to root AP's private VAP using WPS and acquired private VAP's credentials. After successful WPS, STA VAP was reconfigured and the STA VAP connected to backhaul AP VAP on root AP.

With the implementation of the functionality to clone backhaul VAP's credentials on the repeater AP or RE, on root AP, backhaul AP VAPs are not dependent on private VAP's credentials. By default, all backhaul AP VAPs that are newly created obtain the following credentials:

- SSID: Backhaul
- Encryption: psk2+ccmp
- Key: 1234567890

These credentials can be overridden by user in the following two ways:

- By editing the /etc/config/wireless configuration file.
- By specifying the credentials in repacd config file.

**NOTE** Credentials provided in repacd config file have highest precedence. If no credentials in repacd are provided, then credentials provided by user in /etc/config/wireless file are considered. Initially, when VAPs are created, default configuration is applied.

Backhaul AP VAPs are enabled with WPS. Total WPS duration is three minutes. WPS duration is shared between private VAPs and backhaul VAPs. The total WPS duration is split as follows

- The initial one minute is used by private VAPs.
- The next 1 minute and 30 secs are used by backhaul 5GHz VAP.
- The final 30 secs are used by backhaul 2.4 GHz VAP.

**NOTE** When WPS is enabled on both private VAPs and backhaul VAPs, the total WPS duration is split into three sessions—0-60 secs for private VAPs, 61s-150 secs for backhaul 5GHz VAP, and 151s-180 secs for backhaul 2.4GHz VAP. As a result, the WPS duration for private VAPs is reduced to 60 secs, and a delay of 60 secs is observed before backhaul VAPs are connected.

On repeater, STA VAP (by default part of backhaul network) is connected to backhaul AP VAP on root AP using WPS. This is achieved using SON REPT IEs; association is denied if AP has the IE and STA does not have the IE, or if the AP does not have the IE but the STA has the IE.

After connection is successful, using extended WPS enhancement feature, the credentials obtained by STA VAP are propagated to other backhaul STA and AP VAPs. For this behavior, the following two changes are made to the WPS enhancement for repeater feature:

- Support WPS on multiple VAPs of the same radio
- When WPS completes on STA VAP, use wpa\_cli to configure other STA VAP

Enter the following commands to configure additional global settings for this feature to work properly:

```
uci set wireless.qcawifi=qcawifi
uci set wireless.qcawifi.wps_pbc_extender_enhance=1
uci set wireless.qcawifi.wps_pbc_overwrite_ap_settings_all=1
uci set wireless.qcawifi.wps_pbc_overwrite_sta_settings_all=1
```

The following are the per-backhaul VAP settings; these settings are automatically defined by repacd:

```
uci set wireless.@wifi-iface[X].wps_pbc_enable=1
uci set wireless.@wifi-iface[X].wps_pbc_start_time=<appropriate start time>
uci set wireless.@wifi-iface[X].wps_pbc_duration=<appropriate duration>
uci set wireless.@wifi-iface[X].wps_pbc=1
uci set wireless.@wifi-iface[X].wps_pbc_noclone=0
```

The following is the fronthaul VAP configuration; this setting is automatically defined by repacd:

```
uci set wireless.@wifi-iface[X].wps_pbc_noclone=1
```

Some of the preceding configuration might need to be applied manually in case of tri-radio configuration.

## 19.19 Wi-Fi SON API for lbd blacklist

This section describes the design for marking clients as permanently steering-unfriendly based on MAC address. A configuration mechanism to mark clients as permanently steering-unfriendly based on MAC address and a debug CLI mechanism to modify whether a client is considered permanently unfriendly or not are implemented. After clients are marked as permanently steering unfriendly, they are no longer steered. The default behavior is steering-friendly and not permanently steering-unfriendly.

To differentiate "steering unfriendly" with this newly introduced "permanently steering-unfriendly", in this design, "disabling/enabling steering" is used instead of "permanently steering-unfriendly/friendly".

A flag is added to stadb entry to indicate disabling and enabling of steering as follows:

```
typedef struct stadbEntryPriv_t {
    /// Whether the device supports MU-MIMO
    LBD_BOOL isMUMIMOSupported : 1;

    /// whether there is a global prohibition against steering for this
    client
    LBD_BOOL isSteeringDisallowed: 1;

    /// Timestamp of the last time the entry was updated.
    time_t lastUpdateSecs;
```

The following flag handler is added:

```
/**
```

```

@brief STA allow/disallow AP/BAND steering from the debug CLI
*
@param [in] context the output context
@param [in] cmd the command in the debug CLI
*/
void stadbMenuNoSteerHandler(struct cmdContext *context, const char
*cmd);

```

The following flag reading API is added:

```

/**
@brief Obtain whether the STA is prohibited from steering under
      all conditions by debug CLI control
*
@param [in] handle the given BSS stat entry to compare
@param [in] bssStat2 the given BSS stat entry to compare with
*
@return LBD_TRUE if BSS1 is older, otherwise return LBD_FALSE
*/
LBD_BOOL stadbEntry_isSteeringDisallowed(const stadbEntry_handle_t entry)

```

The “nosteer” parameter is added to stadb menu, link to the handler and help updated as follows:

```

static const char *stadbMenuNoSteerHelp[] = {
    " nosteer -- control whether steering is disallowed for a STA",
    "Usage:",
    "\tnosteer <mac_addr> <1|0>: disallow/allow steering (respectively)"
for <mac_addr>,
    NULL
};

static const struct cmdMenuItem stadbMen
{
    "d", stadbMenuDebugHandler, NULL, stadbMenuDebugHelp },
    {"rss", stadbMenuRSSIHandler, NULL, stadbMenuRSSIHelp },
    {"act", stadbMenuActivityHandler, NULL, stadbMenuActivityHelp },
    {"nosteer", stadbMenuNoSteerHandler, NULL, stadbMenuNoSteerHelp },
    {"diaglog", stadbMenuDiaglogHandler, NULL, stadbMenuDiaglogHelp },
    CMD_MENU_END()
};

```

The nosteer command controls whether steering is disallowed for a STA

Usage:

```
nosteer <mac_addr> <1|0>: disallow/allow steering (respectively) for <mac_addr>
```

The stadb print summary is updated. The following is a sample output:

```

@stadb nosteer BC:4C:C4:8B:3D:4A 1
@stadb s out
Num entries = 15

```

MAC Address	Age	Bands	
E0:9D:31:09:C2:88	52	5	Steer Allowed
02:03:7F:87:88:59	6	5	Steer Allowed

8C:70:5A:C3:EE:A0	32	5	Steer Allowed
08:11:96:3F:16:E4	6	5	Steer Allowed
3C:A9:F4:1D:FC:D0	83	5	Steer Allowed
AC:BC:32:94:20:CF	1	25	Steer Allowed
00:03:7F:40:11:4F	1	25	Steer Allowed
A0:88:B4:C7:0B:2C	26	5	Steer Allowed
92:FD:F0:02:06:11	4	2	Steer Allowed
92:FD:F0:02:0A:25	4	5	Steer Allowed
48:51:B7:16:B5:B6	35	5	Steer Allowed
BC:4C:C4:8B:3D:4A	3		Steer Disallowed
00:24:D7:DF:54:AC	27	25	Steer Allowed
A0:88:B4:D1:7A:D0	20	5	Steer Allowed
CA:FF:28:CD:C9:E5	0	5	Steer Allowed

The flag checking in critical path is added as follows:

```

steerexecImplCmnSteeringAcceptType_e steerexecImplCmnSteerOK
    return steerexecImplCmnSteeringAcceptType_rejectProhibited;
}

+ //if STA steering flag is set to true, then cannot to be steered this
STA.
if (stadbEntry_isSteeringDisallowed(entry)){
    return steerexecImplCmnSteeringAcceptType_rejectProhibited;
}

if (steerexecImplCmnIsLegacySteer(steerType)) {
// For legacy steering - just make sure not marked as unfriendly
if (state->legacy.steeringUnfriendly &&

static steerexecImplCmnSteeringType_e
steerexecImplCmnDetermineSteeringType

    steerexecImplCmnSteeringType_e steerType;

//if STA steering flag is set to true, then cannot to be steered this
STA.
if (stadbEntry_isSteeringDisallowed(entry)){
    return steerexecImplCmnSteeringType_none;
}

// Steering unfriendly devices cannot be steered until a timer
expires
// that lets us try steering them again.
if (state->legacy.steeringUnfriendly) {

```

## Limitations

- Currently band steering does not persist any lbd client info across reboot. So persistence option of “permanently steering unfriendly” (disabling steering) is not supported now.
- User must save/implementation the list by themselves and when re-boot, they need to set it again by the command same as they save their own white/black list in their GUI s

- Also, the user must monitor the lbd or hyd to make sure does it restart by the different PID. If PID is changed, then they need to re-set it again.
- Nosteer setting will not be cloned from AP to RE, customer needs to manually add nosteer setting to RE or add it to their customized AP cloning protocol.

### 19.19.1 Sample configuration scenarios

The following sample test scenarios describe the disabling of band steering in pre-association and post-association steering conditions with overload condition detected:

#### Disabling steering in single AP band-steering

AP is configured in DBDC mode and wireless client should be dual band clients.

1 Start the APUT with both bands enabled (DBDC mode)

2 Enable the band steering on the APUT using the following commands

```
/etc/init.d/lbd stop
uci set lbd.@config[0].Enable=1
uci commit
/etc/init.d/lbd start
```

3 Check band steering(lbd) daemon is running using the command "ps | grep lbd" lbd daemon is running

4 Open a telnet session to the APUT on port no.7787 for the Band Steering daemon debug logs. And

Enable logs using the following commands:

```
dbg here
dbg level probe dump
dbg level stamon dump
dbg level bandmon dump
```

**NOTE** Open a Telnet session to the APUT on port no.7787 for the Band Steering daemon (lbd) debug logs. Open a Telnet session to the APUT on port no.7777 for the -Hy-Fi daemon (hyd) debug logs.

5 Connect two wireless clients say STA1 and STA2 to the 2.4 GHz band and run ping to check the connectivity between them. STA1 and STA2 are successfully connected to 2.4 GHz band (Band 0) of APUT

6 Check if disabling steering is added to stadb using the following command:

```
stadb nosteer
```

The nosteer parameter controls whether steering is disallowed for a STA. The usage of this parameter is as follows:

```
nosteer <mac_addr> <1|0>; disallow/allow steering (respectively) for <mac_addr>
```

7 Check the default settings of disabling steering flag using the `stadb s` command.

- STA1 and STA2 are marked as “Steer Allowed”
- 8 Check if disabling steering flag can be set/get using the following command:  
`stadb nosteer <STA1_MAC> 1  
stadb s`  
 STA1 is marked as “Steer Disallowed”, STA2 is marked as “Steer allowed”
- 9 Disconnect STA1 and STA2STA1 and STA2 are disconnected
- 10 Check if disabling steering flag is saved after STA disassociated using the following cmd:  
`stadb s`  
 STA1 is marked as “Steer Disallowed”, STA2 is marked as “Steer Allowed”

### **Disabling steering in pre-association band steering with overload detection in 5GHz**

AP is configured in DBDC mode and wireless client should be dual band clients.

- 1 Start the APUT with both bands enabled (DBDC mode)
- 2 Enable the band steering on the APUT using the following commands  
`/etc/init.d/lbd stop  
uci set lbd.@config[0].Enable=1  
uci commit  
/etc/init.d/lbd start`
- 3 Check band steering(lbd) daemon is running using the command "ps | grep lbd"lbd daemon is running
- 4 Open a telnet session to the APUT on port no.7787 for the Band Steering daemon debug logs. Enable logs using the following commands:  
`dbg here  
dbg level probe dump  
dbg level stamon dump  
dbg level bandmon dump`
- 5 Connect two wireless clients say STA1 and STA2 to the 2.4 GHz band and run ping to check the connectivity between them.STA1 and STA2 are successfully connected to 2.4 GHz band (Band 0) of APUT
- 6 Mark STA1 and STA2 as enabling steering using the following command:  
`stadb nosteer <STA1_MAC> 0  
stadb nosteer <STA2_MAC> 0  
stadb s`  
 STA1 and STA2 are marked as “Steer Enable”
- 7 Disconnect STA2STA2 is disconnected
- 8 Inject traffic onto 2.4GHz at t0 using iperf/chariot running between STA1 and the AP so that 2.4 GHz band is overloaded.Traffic from AP to STA1 runs successfully.

9 Verify AP detects that the 2.4GHz is overloaded no later than t0 + overload\_detection\_time + delta1 (from debug log msgs from the AP).

“Bandmon s” command output in the telnet session confirms 2.4 GHz band is overloaded

10 Move the test wireless client STA2 close to the AP so that 5 GHz signal is strong, ensure 5 GHz RSSI > (5 GHz RSSISteeringPoint i.e. 30 dB) and turn on the Wi-Fi.

“stadb s” command output confirms 5 GHz for STA2 is greater than 30 dB

11 Connect STA2 to the AP. In the sniffer trace verify that when client sends Authentication frame on 2.4 GHz, check that AP rejects it and client tries and successfully associates with 5 GHz.

12 Confirm the pre-association band steering from telnet session debug logs. Check for

i) “Tx’ed Auth reject...” message for Authentication reject on 2.4 Ghz (Band 0).

ii) “Client <STA2 MAC addr> associated on 5 Ghz (Band 1)

iii) “Steering <STA2 MAC addr> to band 1 is complete”

13 Mark STA2 as disabling steering using cmd:

stadb nosteer <STA2\_MAC> 1

stadb s.

STA1 is marked as “Steer Allowed”, STA2 is marked as “Steer Disallowed”

14 Disconnect STA2.

STA2 is disconnected

15 Inject traffic onto 2.4GHz at t0 using iperf/chariot running between STA1 and the AP so that 2.4 GHz band is overloaded.

Traffic from AP to STA1 runs successfully.

16 Verify AP detects that the 2.4GHz is overloaded no later than t0 + overload\_detection\_time + delta1 (from debug log msgs from the AP).

The Bandmon s command output in the telnet session confirms 2.4 GHz band is overloaded

17 Move the test wireless client STA2 close to the AP so that 5 GHz signal is strong, ensure 5 GHz RSSI > (5 GHz RSSISteeringPoint i.e. 30 dB) and turn on the WiFi “stadb s” command output confirms 5 GHz for STA2 is greater than 30 dB

18 Connect STA2 to the AP.

In the sniffer trace verify that when client sends Authentication frame on 2.4 GHz, check that AP does not reject it and client successfully associates with 2 GHz.

19 Confirm the pre-association band steering does not happen from telnet session debug logs. No “Tx’ed auth reject...” message from AP

## Disabling steering in pre-association band steering with overload detection in 5GHz

AP is configured in DBDC mode and wireless client should be dual band clients.

- 1 Start the APUT with both bands enabled (DBDC mode)
- 2 Enable the band steering on the APUT using the following commands  

```
/etc/init.d/lbd stop
uci set lbd.@config[0].Enable=1
uci commit
/etc/init.d/lbd start
```
- 3 Check band steering (lbd) daemon is running using the command "ps | grep lbd".  
 lbd daemon is running
- 4 Open a telnet session to the APUT on port no.7787 for the Band Steering daemon debug logs. Enable logs using the following commands:

```
dbg here
dbg level probe dump
dbg level stamon dump
dbg level bandmon dump
```

- 5 Connect two wireless clients say STA1 and STA2 to the 5 GHz band and run ping to check the connectivity between them.

STA1 and STA2 are successfully connected to 5 GHz band (Band 1) of APUT

- 6 Mark STA1 and STA2 as enabling steering using the following command:

```
stadb b s
```

STA1 and STA2 are marked as "Steer Allowed"

- 7 Disconnect STA2STA2 is disconnected

- 8 Inject traffic onto 5 GHz at t0 using iperf/chariot running between STA1 and the AP so that 5 GHz band is overloaded.

Traffic from AP to STA1 runs successfully.

- 9 Verify AP detects that the 5 GHz is overloaded no later than  $t_0 + \text{overload\_detection\_time} + \Delta t$  (from debug log msgs from the AP).

The "Bandmon s" command output in the telnet session confirms 5 GHz band is overloaded

- 10 Move the test wireless client STA2 close to the AP so that 2.4 GHz signal is strong and turn on the WIFI.

The "stadb s" command output confirms 2.4 GHz for STA2 is greater than 5 GHz

- 11 Connect STA2 to the APUT.

In the sniffer trace verify that when client sends Authentication frame on 5 GHz, check that AP rejects it and client tries and successfully associates with 2.4 GHz.

- 12 Confirm the pre-association band steering from telnet session debug logs. Check for

- i) “Tx'ed Auth reject...” message for Authentication reject on 5 Ghz (Band 0).
  - ii) “Client <STA2 MAC addr> associated on 2.4 Ghz (Band 0)
  - iii) “Steering <STA2 MAC addr> to band 0 is complete”
- 13     Mark STA2 as disabling steering using the following commands:
- ```
stadb nosteer <STA2_MAC> 1
stadb s
```
- STA1 is marked as “Steer Allowed”, STA2 is marked as “Steer Disallowed”
- 14     Disconnect STA2STA2 is disconnected
- 15     Inject traffic onto 5 GHz at t0 using iperf/chariot running between STA1 and the AP so that 5 GHz band is overloaded.Traffic from AP to STA1 runs successfully.
- 16     Verify AP detects that the 5 GHz is overloaded no later than  $t_0 + \text{overload\_detection\_time} + \Delta t_1$  (from debug log msgs from the AP)  
“Bandmon s” command output in the telnet session confirms 2.4 GHz band is overloaded
- 17     Move the test wireless client STA2 close to the AP so that 2.4 GHz signal is strong and turn on the Wi-Fi.  
“stadb s” command output confirms 2.4 GHz for STA2 is greater than 5 GHz
- 18     Connect STA2 to the APUT.  
In the sniffer trace verify that when client sends Authentication frame on 5 GHz, check that AP does not reject it and client successfully associates with 5 GHz.
- 19     Confirm the pre-association band steering does not happen from telnet session debug logs.  
No “Tx'ed auth reject...” message from APis received.

### **Disabling steering in post-association band steering with overload detection in 2.4 GHz**

AP is configured in DBDC mode and wireless client should be dual band clients.

- 1     Start the APUT with both bands enabled (DBDC mode)
- 2     Enable the band steering on the APUT using the following commands  

```
/etc/init.d/lbd stop
uci set lbd.@config[0].Enable=1
uci commit
/etc/init.d/lbd start
```
- 3     Check band steering(lbd) daemon is running using the command, `ps | grep lbd`.  
lbd daemon is running
- 4     Open a telnet session to the APUT on port no.7787 for the Band Steering daemon debug logs. Enable logs using the following commands:

```
dbg here
dbg level probe dump
dbg level stamon dump
```

```
dbg level bandmon dump
```

- 5 Connect the three wireless clients say STA1, STA2, and STA3 to the 2.4 GHz band and run ping to check the connectivity between them.

STA1, STA2, and STA3 are successfully connected to 2.4 GHz band (Band 0) of APUT

- 6 Mark STA1 and STA2 as enabling steering, STA3 as disabling steering using the following commands:

```
stadb nosteer <STA1_MAC> 0
stadb nosteer <STA2_MAC> 0
stadb nosteer <STA3_MAC> 1
stadb s
```

STA1, STA2 are marked as “Steer Allowed”, STA3 is marked as “Steer Disallowed”

- 7 Inject traffic onto 2.4 GHz at t0 using iperf/chariot running from AP to STA1, STA2, and STA3 so that 2.4 GHz band is overloaded.

Traffic from AP to STA1, STA2, and STA3 runs successfully.

- 8 Verify AP detects that the 2.4 GHz is overloaded no later than t0 + overload\_detection\_time + delta1 (from debug log messages from the AP)

The “Bandmon s” command output in the telnet session confirms 2.4 GHz band is overloaded.

- 9 Move the test wireless clients STA2 and STA3 close to the AP so that 5 GHz signal is strong and ensure 5 GHz RSSI > (5GHz RSSISteeringPoint i.e. 30 dB).

The “stadb s” command output confirms 5 GHz for STA2 and STA3 is greater than 30 dB.

- 10 Stop traffic for STA2 from APUT at t1 time.

Check the telnet session logs and sniffer trace. In the telnet session debug logs, verify that client STA2 associates with 5 GHz no later than “t1 + Inactivity\_timer\_2\_overload + delta” and successfully associates with 5 GHz.

In the sniffer trace, check the STA2 disconnects on 2.4 GHz and connects on 5 GHz.

- 11 Stop traffic for STA3 from APUT at t2 time.

Check the telnet session logs and sniffer trace. In the telnet session debug logs, verify that client STA3 stay with 2.4GHz. In the sniffer trace, check the no disassociation/deauthentication frames from STA3 to 2.4GHz.

### **Disabling steering in post-association band steering with overload detection in 5GHz**

AP is configured in DBDC mode and wireless client should be dual band clients.

- 1 Start the APUT with both bands enabled (DBDC mode)

- 2 Enable the band steering on the APUT using the following commands

```
/etc/init.d/lbd stop
uci set lbd.@config[0].Enable=1
uci commit
```

```
/etc/init.d/lbd start
```

3 Check band steering(lbd) daemon is running using the command "ps | grep lbd" lbd daemon is running

4 Open a telnet session to the APUT on port no.7787 for the Band Steering daemon debug logs. Enable logs using the following commands:

```
dbg here
dbg level probe dump
dbg level stamon dump
dbg level bandmon dump
```

5 Connect the three wireless clients say STA1, STA2, and STA3 to the 5 GHz band and running to check the connectivity between them. STA1, STA2, and STA3 are successfully connected to 5 GHz band (Band 1) of APUT

6 Mark STA1 and STA2 as enabling steering, STA3 as disabling steering using the following command:

```
stadb nosteer <STA1_MAC> 0
stadb nosteer <STA2_MAC> 0
stadb nosteer <STA3_MAC> 1
stadb s
```

STA1, STA2 are marked as "Steer Allowed", STA3 is marked as "Steer Disallowed"

7 Inject traffic onto 5 GHz at t0 using iperf/chariot running from AP to STA1, STA2, and STA3 so that 5 GHz band is overloaded.

Traffic from AP to STA1, STA2, and STA3 runs successfully.

8 Verify AP detects that the 5 GHz is overloaded no later than  $t_0 + \text{overload\_detection\_time} + \text{delta1}$  (from debug log messages from the AP).

The "Bandmon s" command output in the telnet session confirms 5 GHz band is overloaded.

9 Move the test wireless clients STA2 and STA3 close to the AP so that 2.4 GHz signal is strong, attenuate the 5 GHz signal.

The "stadb s" command output confirms 2.4 GHz RSSI for STA2 and STA3 is greater than 20 dB.

10 Stop traffic for STA2 from APUT at t1 time.

Check the telnet session logs and sniffer trace. In the telnet session debug logs, verify that client STA2 associates with 2.4 GHz no later than " $t_1 + \text{Inactivity\_timer\_5\_overload} + \text{delta}$ " and successfully associates with 2.4 GHz.

In the sniffer trace, check the STA2 disconnects on 5 GHz and connects on 2.4 GHz.

11 Stop traffic for STA3 from APUT at t2 time. Check the telnet session logs and sniffer trace.

In the telnet session debug logs, verify that client STA3 stay with 5 GHz. In the sniffer trace, check the no disassociation/deauthentication frames from STA3 to 5 GHz.

## 19.20 Separate Wi-Fi SON AP steering threshold values for 2.4 GHz and 5 GHz bands

The functionality to configure different AP steering threshold values for 2.4 GHz and 5 GHz bands using UCI commands and the lbd configuration file are introduced. UCI commands and additional parameters per band are implemented to allow configuration of the LowRSSIAPSteerThreshold\_CAP and LowRSSIAPSteerThreshold\_RE HYD parameters separately for 2.4 GHz and 5 GHz.

LowRSSIAPSteerThreshold\_CAP, LowRSSIAPSteerThreshold\_RE are existing non-band specific lbd configuration parameters. These are used only in multi-AP setup. On AP, they are present in default lbd.config file (that is, /etc/config/lbd). They can be set via UCI command or directly in /etc/config/lbd. When hyd service starts, lbd-config.sh script to parse these multi-AP specific parameters and finally stores it in /tmp/hyd.conf file on AP. Hyd service takes values from /tmp/hyd.conf.

Additional band-specific parameters are introduced with \_W2 and \_W5 suffixes, which represent 2.4 GHz and 5 GHz band respectively. The lbd-config.sh file is modified for parsing these parameters. Rule for parsing is that if band-specific AP Steering parameter is present, it will be used, else it will fall back on non-band generic AP Steering parameter value.

**NOTE** These new options are not present in default lbd config file (/etc/config/lbd). User needs to configure them explicitly.

The following UCI options are added to configure AP Steering thresholds for 2.4 GHz and 5 GHz bands. The \_W2 and \_W5 suffixes in these UCI options represent 2.4 GHz and 5 GHz bands respectively.

- LowRSSIAPSteerThreshold\_CAP\_W2
- LowRSSIAPSteerThreshold\_RE\_W2
- LowRSSIAPSteerThreshold\_CAP\_W5
- LowRSSIAPSteerThreshold\_RE\_W5

Original non-band specific options LowRSSIAPSteerThreshold\_CAP, LowRSSIAPSteerThreshold\_RE still exists and remains in default lbd config file.

When band-specific AP Steering threshold is present, it will be used, else non-band specific AP Steering parameter value will be used. Band specific AP Steering values can be either set via UCI commands or directly in /etc/config/lbd file.

To configure the band-specific AP steering values using CLI commands:

```
uci set lbd.APSteer.LowRSSIAPSteerThreshold_CAP_W2=25
uci set lbd.APSteer.LowRSSIAPSteerThreshold_CAP_W5=55
uci set lbd.APSteer. LowRSSIAPSteerThreshold_RE_W2=25
uci set lbd.APSteer. LowRSSIAPSteerThreshold_RE_W5=55
```

To configure the band-specific AP steering values using the **/etc/config/lbd** file:

```
config APSteer 'APSteer'
option LowRSSIAPSteerThreshold_CAP_W2 '25'
option LowRSSIAPSteerThreshold_CAP_W5 '55'
option LowRSSIAPSteerThreshold_RE_W2 '25'
```

```
option LowRSSIAPSteerThreshold_RE_W5 '55'
```

To set up a multi-AP and range extender and enter the band-specific AP steering parameters using the UCI commands, do the following:

1. On CAP, enable 2G and 5G VAPs, enable radio resource measurement (RRM), and wireless distribution system (WDS).

```
uci set wireless.wifi-iface[0].wds=1
uci set wireless.wifi-iface[0].rrm=1
uci set wireless.wifi-iface[1].wds=1
uci set wireless.wifi-iface[1].rrm=1
uci commit wireless
```

2. Bring up the wireless interfaces using the wifi command.

3. Disable mcsd (as it cannot run at the same time as hyd):

```
uci set mcsd.config.Enable=0
uci commit mcsd
/etc/init.d/mcsd stop
```

4. Enable and start hyd:

```
uci set hyd.@config[0].Enable=1
uci commit hyd
/etc/init.d/hyd start
```

5. Configure the range extender (RE) in a similar manner.

6. To validate the UCI options for low RSSI AP steering threshold for central AP and RE, enter the following UCI commands or add new band-specific parameters directly in /etc/config/lbd file:

```
uci set lbd.APSteer.LowRSSIAPSteerThreshold_CAP_W2=25
uci set lbd.APSteer.LowRSSIAPSteerThreshold_CAP_W5=55
uci set lbd.APSteer.LowRSSIAPSteerThreshold_RE_W2=25
uci set lbd.APSteer.LowRSSIAPSteerThreshold_RE_W5=55
uci commit
```

7. Restart hyd service by entering the following command:

```
root@OpenWrt:~# /etc/init.d/hyd start
net.bridge.bridge-nf-call-custom = 1
hyd: restarting daemon
```

New values for 2 GHz and 5 GHz are reflected in /tmp/hyd.conf in [WLANIF2G] and [WLANIF5G] respectively as shown in the following portion of the output:

#### **NOTE**

- The hyd service uses values from /tmp/hyd.conf, so we are checking final values from this file.
- LowRSSIAPSteeringThreshold is the name used by hyd for AP steering threshold.

```
[WLANIF2G]
InterferenceDetectionEnable=1
InactIdleThreshold=10
InactOverloadThreshold=10
InactCheckInterval=1
InactRSSIXingHighThreshold=35
LowRSSIXingThreshold=10
BcnrptActiveDuration=50
BcnrptPassiveDuration=200
HighTxRateXingThreshold=50000
HighRateRSSIXingThreshold=30
LowRSSIAPSteeringThreshold=25

[WLANIF5G]
InterferenceDetectionEnable=1
InactIdleThreshold=10
InactOverloadThreshold=10
InactCheckInterval=1
InactRSSIXingHighThreshold=20
InactRSSIXingLowThreshold=0
LowRSSIXingThreshold=10
BcnrptActiveDuration=50
BcnrptPassiveDuration=200
LowTxRateXingThreshold=6000
LowRateRSSIXingThreshold=0
LowRSSIAPSteeringThreshold=55
```

## 19.21 Best uplink algorithm for PLC interface metrics in a daisy chain Wi-Fi SON and PLC network

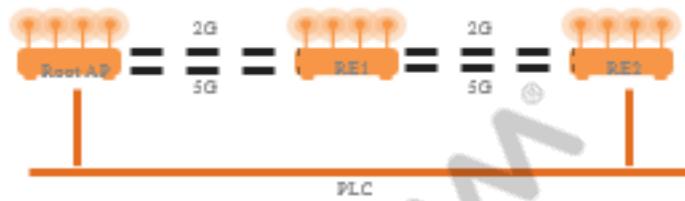
The capability to apply the best uplink algorithm to use PLC interface link metrics in a Wi-Fi SON network integrated with PLC is introduced. With daisy chain topologies involving a CAP, RE1, and RE2 (where RE1 can either be an REH172 device or a DK01/DK07 device), RE2 can reach the CAP (gateway) through different upstream devices with different backhaul interfaces. When multiple paths are available to reach the CAP, a potential loop can occur in the Wi-Fi SON with PLC network. As a result, the following problems are noticed in the Wi-Fi SON network

- Loop observed on all nodes with broadcast packets
- Redundant MAC learning on multiple SON nodes
- Ping instability between repeaters and Ethernet clients
- Slowness in the network

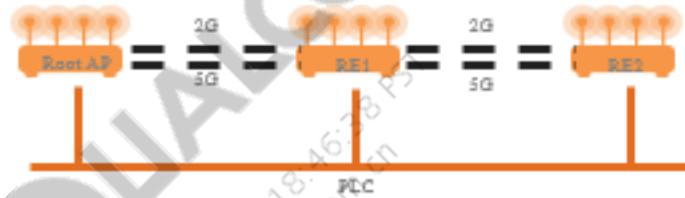
To avoid the complexity in daisy chain topologies when different backhaul interfaces connect to different upstream devices, the recommendation is to select only one backhaul interface from second hop onwards. The selection of backhaul interface is performed using the best uplink selection algorithm applying the PLC interface link metrics.

Only one backhaul interface, when found in a daisy chain topology, is selected. In a Wi-Fi SON network integrated with PLC that contains daisy chain topology support, a possibility of the second-hop repeater reaching the CAP through two different backhaul interfaces (WLAN/PLC) exists. The capability to apply the best uplink selection algorithm is implemented in a daisy chain topology that contains the *CAP ---- RE1 ---- RE2* connection.

The feature supports the first level of daisy chain. The following figures illustrate a simple first level of daisy-chain network, where RE1 is connected to the CAP and RE2 is connected to RE1 along with PLC interfaces. Both REs have HYD instances running, and AP and band steering occurs in the usual manner across them.



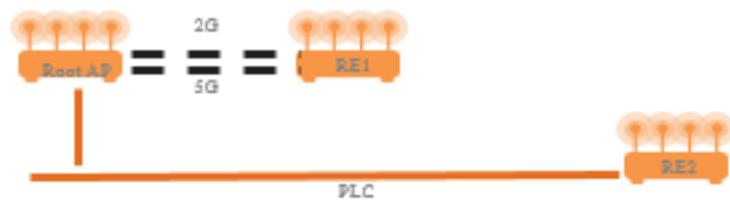
**Figure 19-10 PLC daisy chain with RE1 as DK01/DK07**



**Figure 19-11 PLC daisy chain with RE1 as REH172**

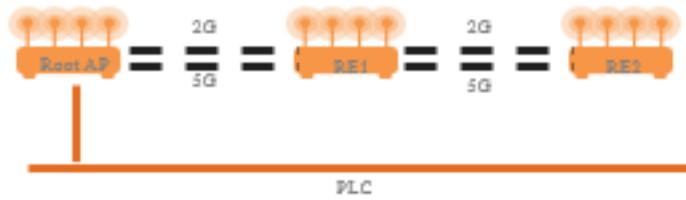
The RE2 can reach CAP through WLAN and PLC two different backhaul links. The repacd best-uplink algorithm running on RE2 must consider PLC Link Metrics also to select the best path to reach CAP. After selecting the best path, the preceding daisy chain topologies will become the following topologies:

With RE1 as DK01/DK07, if PLC interface is selected as backhaul, then the topology becomes a star topology as shown in the following figure. The daisy chain topology becomes star topology with one repeater connected with WLAN and another repeater connected with PLC.



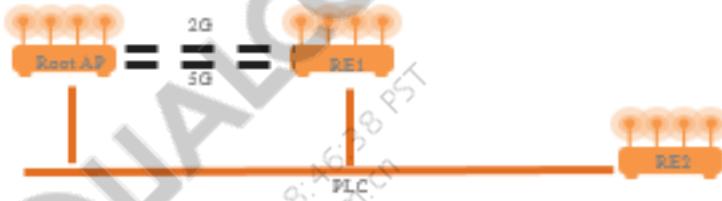
**Figure 19-12 PLC daisy chain becomes a star topology after selecting PLC as backhaul in Daisy Chain topology**

With RE1 as REH172, if WLAN interface is selected as backhaul, then the topology becomes a pure WLAN daisy chain topology as shown in the following figure:



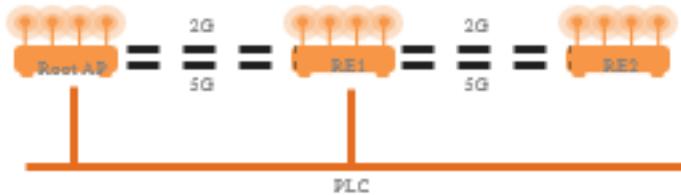
**Figure 19-13 PLC daisy chain becomes pure WLAN daisy chain topology for CAP-RE1-RE2 after selecting WLAN interface as backhaul**

With RE1 as REH172, if PLC interface is selected as backhaul, then the topology becomes a star topology as shown in the following figure. The daisy chain topology becomes star topology with one repeater connected with WLAN and PLC, and another repeater connected only with PLC.



**Figure 19-14 PLC daisy chain becomes a star topology after selecting PLC as backhaul in Daisy Chain topology**

With RE1 as REH172, if PLC interface is selected as backhaul, then the topology becomes a pure WLAN daisy chain topology as shown in the following figure:



**Figure 19-15 PLC daisy chain becomes pure WLAN daisy chain topology for CAP-RE1-RE2 after selecting WLAN interface as backhaul**

Set the following parameters in the repacd configuration file,:

```
uci set repacd.BackhaulMgr.SelectOneBackHaulInterfaceInDaisy='1'
uci set repacd.BackhaulMgr.BackHaulMgrRateNumMeasurements='10'
uci set repacd.BackhaulMgr.PLCLinkThresholdto2G='60'
uci set repacd.BackhaulMgr.SwitchInterfaceAfterCAPPINGTimeouts='10'
uci commit repacd
uci commit
```

- SelectOneBackHaulInterfaceInDaisy is enable the feature to select only one backhaul interface when found WLAN and PLC.

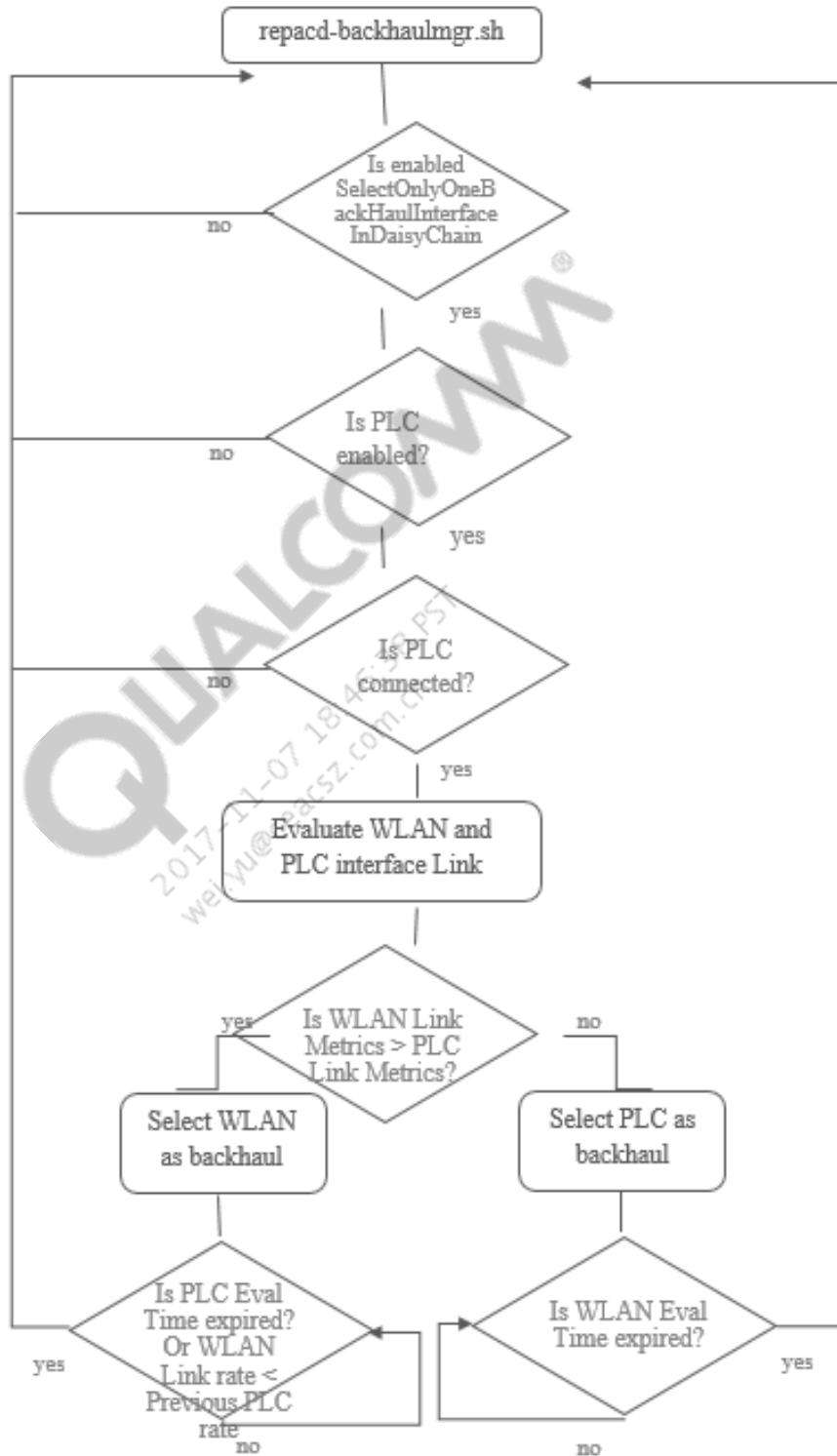
- BackHaulMgrRateNumMeasurements is used to consider number of samples to measure before selecting a backhaul interface, either WLAN or PLC.
- PLCLinkThresholdto2G is used to select PLC interface as backhaul when WLAN 5G link rate is zero or 5G is not connected to its upstream and 2G is connected to upstream. In such situation if PLC link rate is less than configured threshold value then WLAN 2G interface is selected, otherwise if PLC link rate is greater than configured threshold value then PLC interface is selected as backhaul.
- SwitchInterfaceAfterCAPPINGTimeouts is used to consider number of ping outs to switch interface from PLC to WLAN, when CAP is not reachable thru PLC interface.

In cases of failover scenarios such as the CAP not being reachable through PLC interface, then the expected switchover time for traffic depends on the following:

- Bring up of WLAN (2G and 5G) interfaces
- Association of interfaces

The flowchart of the design is as follows:

QUALCOMM  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn



A new file, `repacd-plcmon.sh`, is created to verify whether PLC is configured, connected, and to get the PLC link metrics. A new file, `repacd-backhaulmgr.sh`, is created to select best uplink at any

point in time. The repacd\_backhaulmgrmon\_check function monitors WLAN and PLC Link rate continuously and selects only one backhaul; the other backhaul is brought down.

1. A verification is performed to determine whether the `SelectOnlyOneBackHaulInterfaceInDaisyChain` option is enabled in the `repacd-backhaulmgr.sh` file. If this option is enabled, the next step is performed.
2. A check is performed to determine whether PLC is enabled and connected. If PLC is enabled and connected, the WLAN and PLC interface link metrics are evaluated.
3. If the WLAN interface link metric is greater than the PLC interface link metric, WLAN interface is selected as the backhaul interface.
4. A check is performed to determine whether the PLC evaluation timer has expired, or whether the WLAN Link rate is less than the previous PLC rate. Either of the following operations is performed, based on the condition that is satisfied:
  - If either of these conditions is satisfied, the WLAN and PLC interface link-metric check is performed to determine the greater metric.
  - If none of these conditions are satisfied, the check is repeated for the PLC evaluation timer and WLAN link rates.
5. If the WLAN interface link metric is less than the PLC interface link metric, PLC interface is selected as the backhaul interface.
6. A check is performed to determine whether the WLAN evaluation timer has expired. Either of the following operations is performed based on whether the condition is satisfied:
  - If so, the WLAN and PLC interface link-metric check to determine which metric is greater is performed.
  - Else, the check is repeated until the WLAN evaluation timer expires.

## Sample configuration topology

Consider a daisy-chain topology that comprises a CAP, RE1 (either an REH172 or a DK01/DK07), and RE2 with two UDP streams of 20 Mbps each. Unidirectional (downstream) traffic flows from CAP to RE2 through the PLC interface. Bring up WLAN interfaces. The repacd selects WLAN as backhaul interface and brings down the PLC interface. The traffic switches from PLC to WLAN in 5-10 seconds.

Bring down WLAN interfaces. The unidirectional traffic from WLAN to PLC takes approximately five minutes to switch over.

When bidirectional (downstream) traffic flows from CAP to RE2 through PLC, bring up WLAN interfaces. The repacd selects WLAN as backhaul interface, and brings down the PLC interface. The traffic switches from PLC to WLAN within 5-10 seconds.

Bring down the WLAN interface. The bidirectional traffic from WLAN to PLC takes approximately five minutes to switch over.

## Limitations

It is a known behavior that the WLAN to PLC traffic switchover takes time. This is the current system behavior because the H-Active/H-Default and other tables are not cleaned up immediately,

whenever any disturbance/fluctuation over the air is observed. The assumption is that WLAN clients or repeater might reconnect immediately.

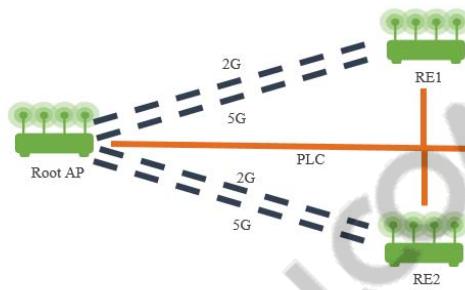
Switching the topology from daisy-chain to star or vice-versa might cause a loss of ping packets between Wi-Fi SON nodes (or repeaters only to internal SON network) until the ageout time period is exceeded. An intermediate node (RE1) that performs automatic path selection to select the end-to-end traffic is risky.

|             |             |                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |    |
|-------------|-------------|-------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| BackhaulMgr | BackhaulMgr | SelectOneBackHaulInterfaceInDaisy               | <p>Set the backhaul interface to be selected in the daisy-chain topology as 1. The repacd-plcmon.sh is used to verify whether the PLC link is configured and connected, and to obtain the PLC Link Metrics.</p> <p>To select the best uplink at a any point in time, the repacd_backhaulmgrmon_check function in the repacd-backhaulmgr.sh file monitors WLAN and PLC Link rate continuously and selects only one backhaul. A verification is performed to determine whether the SelectOnlyOneBackHaulInterfaceInDaisyChain option is enabled in the repacd-backhaulmgr.sh file. If this option is enabled, the next step of checking whether PLC link is enabled is performed. To avoid the complexity in daisy chain topologies when different backhaul interfaces connect to different upstream devices, the recommendation is to select only one backhaul interface from second hop onwards.</p> <p>The selection of backhaul interface is performed using the best uplink selection algorithm applying the PLC interface link metrics. Only one backhaul interface, when found in a daisy chain topology, is selected.</p> | 1  |
| BackhaulMgr | BackhaulMgr | BackHaulMgrRateNumMeasurements                  | <p>Set the number of measurements for the backhaul manager rate as 10. A check is performed to determine whether the PLC evaluation timer has expired, or whether the WLAN Link rate is less than the previous PLC rate.</p> <p>If either of these conditions is satisfied, the WLAN and PLC interface link-metric check is performed to determine the greater metric. If none of these conditions are satisfied, the check is repeated for the PLC eval timer and WLAN link rates.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | 10 |
| PLCLink     | PLCLink     | BackhaulMgr.PLCLinkThresholdto2G                | Select PLC interface as backhaul when WLAN 5G link rate is zero or 5G is not connected to its upstream and 2G is connected to upstream. In such situation if PLC link rate is less than configured threshold value then WLAN 2G interface is selected, otherwise if PLC link rate is greater than configured threshold value then PLC interface is selected as backhaul.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 60 |
| BackhaulMgr | BackhaulMgr | BackhaulMgr.SwitchInterfaceAfterCAPPingTimeouts | Consider number of ping outs to switch interface from PLC to WLAN, when CAP is not reachable through PLC interface.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 10 |

## 19.22 Interoperation of different PLC chipset models in a Wi-Fi SON network

In a Wi-Fi SON network integrated with PLC, support is implemented for a combination of different PLC devices, such as AR742x and QCA75xx chipsets, to be connected with CAP and RE1 respectively. When the CAP is connected with AR742x device and the RE1 is connected with QCA75xx device, the metrics for the medium utilization, TCP utilization, and UDP utilization show desired results.

The following figure shows such a topology:



When CAP and RE are connected with different PLC devices such as AR742x and QCA75xx chipsets, the channel information is read using the Vendor specific MME 0xA200. The Channel Info MME is used to obtain medium utilization and channel capacity estimates. The Channel Info Request and Channel Info Confirm MMEs are available.

**Table 19-6 Channel Info Request MME**

| Offset | Size | Field Name | Description                                                                                                                                                               |
|--------|------|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x0000 | 6    | ODA        | Original Destination Address                                                                                                                                              |
| 0x0006 | 6    | OSA        | Original Source Address                                                                                                                                                   |
| 0x000C | 4    | VLAN       | VLAN Tag (optional)                                                                                                                                                       |
| 0x0010 | 2    | MTYPE      | Ethertype (Home Plug AV =0xe188)                                                                                                                                          |
| 0x0012 | 1    | MMV        | MME Version (1)                                                                                                                                                           |
| 0x0013 | 2    | MMTYPE     | 0xA200 (Request)                                                                                                                                                          |
| 0x0015 | 2    | FMI        | Fragmentable MME Information                                                                                                                                              |
| 0x0017 | 3    | OUI        | Qualcomm Atheros OUI (0x00, 0xb0, 0x52)                                                                                                                                   |
| 0x001A | 4    | MME_LEN    | MME Data Length                                                                                                                                                           |
| 0x001E | 2    | REQ_TYPE   | Request Type<br>0x0001 – Channel Data Rate Request<br>0x0002 – Medium Utilization Request<br>0x0003 – RESERVED (Available for other CHANNEL INFO)<br>...           0xFFFF |
| 0x0020 | 4    | RESERVED   | Reserved for later use.                                                                                                                                                   |

**Table 19-7 Channel Info Confirm MME**

| <b>Offset</b> | <b>Size</b> | <b>Field Name</b> | <b>Description</b>                                                                                                   |
|---------------|-------------|-------------------|----------------------------------------------------------------------------------------------------------------------|
| 0x0000        | 6           | ODA               | Original Destination Address                                                                                         |
| 0x0006        | 6           | OSA               | Original Source Address                                                                                              |
| 0x000C        | 4           | VLAN              | VLAN Tag (optional)                                                                                                  |
| 0x0010        | 2           | MTYPE             | Ethertype (Home Plug AV =0xe188)                                                                                     |
| 0x0012        | 1           | MMV               | MME Version (1)                                                                                                      |
| 0x0013        | 2           | MMTYPE            | 0xA201 (Confirm)                                                                                                     |
| 0x0015        | 2           | FMI               | Fragmentable MME Information                                                                                         |
| 0x0017        | 3           | OUI               | Qualcomm Atheros OUI (0x00, 0xb0, 0x52)                                                                              |
| 0x001A        | 1           | STATUS            | MME Status<br>0x00: Success<br>0x01: Invalid Request Type<br>0x02-0x0F: Reserved for Common Status Info              |
| 0x001B        | 1           | RESERVED          | Reserved for later use                                                                                               |
| 0x001C        | 4           | MME_LEN           | MME Data Length                                                                                                      |
| 0x0020        | 2           | REQ_TYPE          | Request Type<br>0x0001 – PHY Data Rate Request<br>0x0002 – Channel Utilization Request<br>0x0003...0xFFFF – RESERVED |
| 0x0022        | 2           | RESERVED          | Reserved for later use                                                                                               |
| 0x0024        | X           | CHANNEL_DATA      | Data Related to Channel Info. Either:<br>CHANNEL_DATA_RATES<br>or<br>CHANNEL_MEDIUM_UTILIZATION                      |

The CHANNEL\_DATA\_RATES format is shown as follows:

**Table 19-8 CHANNEL\_DATA\_RATES**

| <b>Offset</b> | <b>Size</b> | <b>Field Name</b> | <b>Description</b>                                                                                                             |
|---------------|-------------|-------------------|--------------------------------------------------------------------------------------------------------------------------------|
| 0x0000        | 1           | SUBVERSION_NUM    | Subversion number of the response<br>Currently set to 0x00                                                                     |
| 0x0001        | 1           | NumSTAs           | Number of Stations in the network (not including the node sending this MME) Currently, we limit the NumSTAs to a maximum of 60 |
| 0x0002        | 1           | TEI[0]            | Terminal Equipment Identifier of STA[0]                                                                                        |
| 0x0003        | 6           | MAC_ADDR[0]       | MAC Address of STA[0]                                                                                                          |
| 0x0009        | 2           | TX_RATE [0]       | Estimated transmit channel data rate to STA[0]                                                                                 |
| 0x000B        | 6           | RSVD[0]           | Reserved                                                                                                                       |
| 0x0011        | 1           | TEI[1]            | Terminal Equipment Identifier of STA[1]                                                                                        |

|        |    |             |                                                |
|--------|----|-------------|------------------------------------------------|
| 0x0012 | 6  | MAC_ADDR[1] | MAC Address of STA[1]                          |
| 0x0018 | 2  | TX_RATE [1] | Estimated transmit channel data rate to STA[1] |
| 0x001A | 6  | RSVD[1]     | Reserved                                       |
| 0x001F | XX | ...         | ...                                            |

The CHANNEL\_MEDIUM\_UTILIZATION format is shown as follows:

**Table 19-9 CHANNEL\_MEDIUM\_UTILIZATION**

| Offset | Size | Field Name        | Description                                                                                                                                                  |
|--------|------|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x0000 | 1    | SUBVERSION_NUM    | Subversion number of the response<br>Currently set to 0x00                                                                                                   |
| 0x0001 | 4    | TimeStamp         | System timestamp when this response was generated                                                                                                            |
| 0x0005 | 4    | TDMA_Counter      | Time on wire occupied by TDMA traffic (in units of 1.28 usecs since the counter was reset). Note this in the powerline medium utilization only.              |
| 0x0009 | 4    | CAP3_Counter      | Time on wire occupied by CAP 3 CSMA traffic (in units of 1.28 usecs since the counter was reset). Note this in the powerline medium utilization only.        |
| 0x000D | 4    | CAP2_Counter      | Time on wire occupied by CAP 2 CSMA traffic (in units of 1.28 usecs since the counter was reset). Note this in the powerline medium utilization only.        |
| 0x0011 | 4    | CAP1_Counter      | Time on wire occupied by CAP 1 CSMA traffic (in units of 1.28 usecs since the counter was reset). Note this in the powerline medium utilization only.        |
| 0x0015 | 4    | CAP0_Counter      | Time on wire occupied by CAP 0 CSMA traffic (in units of 1.28 usecs since the counter was reset). Note this in the powerline medium utilization only.        |
| 0x0019 | 4    | nonHPAVLN_Counter | Time on wire occupied by non HomePlug AVLN traffic (in units of 1.28 usecs since the counter was reset). Note this in the powerline medium utilization only. |

The PLC host connected in the Wi-Fi SON network requests for the data rates and medium utilization using the vendor specific hybrid MME 0xA200. The PLC firmware receives the MME and sends the request to the station connected in the audio video logical network (AVLN).

The PLC firmware estimates the data rates and medium utilization based on the station connected in the network and sends the response to the PLC host. The PLC host receives the response and analyzes the data rates and medium utilization info to use the best network.

Support is implemented for the link metrics using the hybrid info MME (0xA200) to read the PHY rates and the medium utilization. Also, the PLC host can receive the link metrics information and process the message.

Only Qualcomm PLC chipsets are supported, and non-Qualcomm PLC devices are not supported in this environment.

### Sample configuration scenario

Consider a sample topology in which the CAP and RE are connected with AR742x and QCA75xx devices respectively. Execute the `hyd` and `plc start` commands to estimate the data rates and medium utilization. The results for the medium utilization, and TCP and UDP rates are estimated. The following is the channel estimation summary:

| Mix Nodes between CAP and RE1 | PHY Rate using MME 0xA038 | TCP Link capacity using MME 0xA200 | UDP Link Capacity using MME 0xA200 | Medium Utilization using MME 0xA200 |
|-------------------------------|---------------------------|------------------------------------|------------------------------------|-------------------------------------|
| QCA7500 < - > QCA7520         | Estimated                 | Estimated                          | Estimated                          | Estimated                           |
| QCA7500 < - > QCA7550         | Estimated                 | Estimated                          | Estimated                          | Estimated                           |
| QCA7500 < - > AR7420          | Estimated                 | Not Estimated                      | Not Estimated                      | Not Estimated                       |
| QCA7520 < - > AR7420          | Estimated                 | Not Estimated                      | Not Estimated                      | Not Estimated                       |
| QCA7550 < - > AR7420          | Estimated                 | Not Estimated                      | Not Estimated                      | Not Estimated                       |

Sample configuration for the QCA75xx node:

```
uci set plc.config.AggrLinkRate='0'
uci set hyd.PathChPlc.HostPLCInterfaceSpeed='1000'
uci set hyd.PathChPlc.MaxMediumUtilization='80'
uci set plc.config.Enabled='1'
uci set plc.config.NetworkPassWd='PLC-SON-Daisychain-12345'
uci set plc.config.PlcIfname='eth1'
uci commit plc
uci commit hyd
uci commit
/etc/init.d/plc start
```

Sample configuration for the AR742x node:

```
uci set plc.config.AggrLinkRate='0'
uci set hyd.PathChPlc.HostPLCInterfaceSpeed='500'
uci set hyd.PathChPlc.MaxMediumUtilization='40'
uci set plc.config.Enabled='1'
uci set plc.config.NetworkPassWd='PLC-SON-Daisychain-12345'
uci set plc.config.PlcIfname='eth1'
uci commit plc
uci commit hyd
uci commit
/etc/init.d/plc start
```

## 19.23 Wi-Fi SON: Skip PFS and improve airtime calculation

In a Wi-Fi SON network, steering enhancements are implemented, starting with QCA\_Networking\_2017.SPF.5.0.3. These enhancements comprise the following:

- Optional wait for prepare for steering (PFS) responses
- Improve airtime calculation on steering decision
- Cleanup of the client list

## 19.23.1 Overview of steering features

### 19.23.1.1 Optional wait for PFS responses

The capability to optionally wait for prepare for steering (PFS) responses is introduced. Prior to the implementation of this functionality, before start steering a client from its associated AP to the candidate AP, the associated AP sent the PFS request to all the APs in the whole network, and then waited for the responses from all of the APs before issuing the actual steering command to the client. In some cases, because of many reasons, the sending AP did not receive responses from all of them in a configurable period of time, and the remote APs may not even get the request. In those cases, the system aborted the steering process. Such a method of working caused several difficulties in complicated topologies and steering was not possible if one or more APs were temporarily out of service.

With the implementation of the capability to optionally wait for PFS responses, as long as the sending (associated) AP receives the response from the candidate APs, the steering will start. If the sending AP gets responses from all of the APs in the network within the defined time period, the sending AP issues the steering command; if the sending AP does not receive all of the responses when the timer expires, the sending AP checks whether it receives the responses from the candidate AP, if it does, the steering will start. Otherwise, steering is aborted.

### 19.23.1.2 Improve airtime calculation on steering decision

Also, the enhancement to improve airtime calculation on steering decision is implemented. Before the implementation of this enhancement, while making decisions for steering an associated client to a candidate BSS, either on the same AP or on different AP, on the same channel or on different channel, the associated AP received two numbers: the first was the client's estimated airtime, and the second was the target channel's measured channel utilization. Then, both these numbers were added together to derive the projected channel utilization after steering. If this sum was greater than the safety threshold configured for that band, steering was not performed.

This mechanism worked properly for steering the client to a different channel. However, whenever the client was steered to the same channel but different AP, because the measured channel utilization has included the client's estimated airtime, the airtime was counted twice and the projected channel utilization was overestimated. As a result, the same channel steering was denied, although it must not have been denied.

This was because of the conservative nature of implementation. The enhancement adds a configuration flag in lbd.config file to decide whether the client's estimated airtime needs to be added to the projected channel utilization while making same channel steering decision.

### 19.23.1.3 Clean up client list

Finally, the enhancement to clean up client list is introduced. If a client is roamed or steered away from an AP, it should be removed from the associated device list in the current AP's database and driver.

Because the original AP might not be able to determine whether an associated device might have left from it, the client might not be removed from the list. However, if the device is roamed or steered to an other AP in the network, the new AP will notify the original AP for the association change and the original AP will call its driver to remove that device from its association list and also update the corresponding database. This operation is performed by the new AP to send Topology Notification message.

More than one issue might cause the failure of removing the client device from the previous associated AP's databases after client roamed or steered away from an AP. However, this specified requirement is significantly addressed and implemented.

## 19.23.2 Design and implementation

### 19.23.2.1 Optionally wait for PFS responses

In the earlier design, when a AP decides to steer an associated device to a new AP, it will send PFS to all the other APs on the network. On receiving the request, the other APs takes actions and then responds to the sender of the request. The sender waits for a certain time period for receiving the response from all the APs and then starts the steering. If the sender does not receive a response from all of the APs when timer expires, the sender aborts the steering.

In the enhanced design, starting with QCA\_Networking\_2017.SPF.5.0.3, while sending PFS request, the sender does not only maintain a counter for pending responses, but also maintain a pending candidate list. Both of the pending response counter and the pending candidate list are updated on receiving responses from other APs. Based on the pending response counter, if the sending AP gets the responses from all the other APs before the timer expires, similar to the previous design, the associated AP issues the steering command, and the timer is canceled. If the associated AP does not get all the responses from all of the other APs when timer expires, the pending candidate list is checked to see if all the candidate(s) have responded. The associated AP issues the steering command as long as it receives the responses from all the candidate APs.

In a scenario in which all the operations occur properly and without errors, all the logic is the same as the original design. If the associated AP cannot reach some of the remote AP for PFS, it might still perform the steering, as long as the unresponsive APs are not involved in the steering process.

### 19.23.2.2 Improve airtime calculation on steering decision

This enhancement enables improvement of the steering decision making process when airtime is concerned. In the original design, when same channel steering was intended, it was not assumed that the client's airtime was already included into the current channel's total airtime utilization. As a result, the client's estimated airtime was counted doubly.

To be conservative, a new configuration flag “ApplyEstimatedAireTimeOnSteering” is added in file “lbd.config”. In running environment, the file is located at “/etc/config/lbd”. This flag is set to be 1 by default. If the value of this flag is changed as 0, this enhancement to improve airtime calculation is triggered, which denotes that the client’s estimated airtime is not added to the projected channel utilization when making same channel steering decision.

|        |        |                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |   |
|--------|--------|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|
| config | config | ApplyEstimatedAireTi<br>meOnSteering | This parameter enables improvement of the steering decision making process when airtime is concerned. In the design without this lbd config parameter, when the same channel steering was intended, it was not assumed that the client’s airtime was already included into the current channel’s total airtime utilization. As a result, the client’s estimated airtime was counted doubly.<br><br>To be conservative, the configuration flag “ApplyEstimatedAireTimeOnSteering” is added in file “lbd.config”. In running environment, the file is located at “/etc/config/lbd”. This flag is set to be 1 by default. If the value of this flag is changed as 0, this enhancement to improve airtime calculation is triggered, which denotes that the client’s estimated airtime is not added to the projected channel utilization when making same channel steering decision. | 1 |
|--------|--------|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|

### 19.23.2.3 Clean up client list

The capability to clean up the client list has been verified.

## 19.23.3 Verify the steering enhancements

This section describes the verification methods for the features regarding optional wait for PFS responses, improved airtime calculation, and client list cleanup processes.

### 19.23.3.1 Verify the optional wait for PFS responses

To verify this enhancement by monitoring the debug message, run `hyt / dbg level steermsg=3 / dbg` here on the associated AP, and monitor the log message from “steermsg” module.

A message stating “**Sent prepare for steering via unicast for STA xx:xx:xx:xx:xx:xx**” is displayed.

If the associated AP does not get the responses from all the APs, and even all the candidate APs, the following error message is displayed: **Failed to receive all prepare for steering responses for xx:xx:xx:xx:xx:xx, remaining [n] transaction ID [m]**

If the associated AP does not get the responses from all the APs, and if the associated AP receives the responses from all of the candidate APs, the following debug message is displayed: **Receive steering responses for xx:xx:xx:xx:xx:xx from all candidates. Remaining [n] transaction ID [m]**

Whenever a PFS response is received from another AP, the following message is displayed:  
*Received prepare for steering response from xx:xx:xx:xx:xx:xx for xx:xx:xx:xx:xx:xx  
transaction [m] status [s] (mid [m]); remaining [r]''*

### 19.23.3.2 Verify the improved airtime calculation

There is no direct information from the log to indicate whether the enhancement is effective or not. Check if the flag “ApplyEstimatedAireTimeOnSteering” is defined in file “/etc/config/lbd” or not. Use “hyt / bandmon / s” to verify the measured channel utilization and the expected airtime increase.

### 19.23.3.3 Verify the client list cleaning process

When a client steered or roamed away from a AP, always check if the client is still in the original AP’s associated device list or not by using “hyt / td / s2”. Make sure the device is only listed in one AP. If the client device is disconnected from the network, it may not be removed from the previous associated AP right away, but it should be not be listed on more than one AP.

## 19.24 Support for REH172 as a root AP

Starting with QCA\_Networking\_2017.SPF.5.0.3, in a Wi-Fi SON network integrated with PLC, REH172 (PQ4018/IPQ4028 + QCA7500) can be configured as a root AP in daisy-chain and star topologies, in addition to the support for REH172 as a repeater AP (range extender) that existed in previous releases.

### 19.24.1 REH172 overview

REH172 builds with IPQ40x8 system-on-chip (SoC) stacked up on top of QCA7500 board. QCA7500 board provides the power supply to Host (IPQ40x8) board. The IPQ40x8 is a highly-integrated system-on-chip (SoC) designed for high-performance, power efficient, and cost-effective 2×2, 802.11ac, dual-band concurrent access-point applications. The SoC incorporates a quad-core ARM Cortex-A7 processor, two dual-band, concurrent 802.11ac Wave-2 Wi-Fi subsystems, and a five-port Gigabit Ethernet Layer2/3/4 multilayer switch supporting line rate network address translation (NAT). It supports one USB3.0 and one USB2.0.

The IPQ40x8 SoC also supports miscellaneous interfaces such as I2S, SPDIF, I2C, SMI, UART, and JTAG, which can be configured as general purpose input/output pins (GPIOs).

Both static NAT and NAPT functions are supported. The HNAT module is built into the SSDK kernel module. When the SSDK module is loaded, the HNAT function is disabled by default.

HNAT supports the following two modes:

- When mode 0 is enabled, it uses full 1 K NPAT entries, but it does not synchronize corresponding counters to Linux kernel.
- When mode 1 is enabled, it synchronizes the counters to Linux kernel but supports only 8 NAPT entries. By default, mode 0 is enabled.

### 19.24.1.1 List of acronyms

| Acronym    | Description                                                                                             |
|------------|---------------------------------------------------------------------------------------------------------|
| EDMA       | Ethernet Direct Memory Access                                                                           |
| ESS        | Ethernet Switch Subsystem                                                                               |
| GPIO       | General-Purpose I/O                                                                                     |
| HLOS       | High-Level Operating System                                                                             |
| IRQ        | Interrupt Request                                                                                       |
| WSS        | Wi-Fi Sub System                                                                                        |
| PBL        | Primary Boot Loader                                                                                     |
| QRFS       | Qualcomm® Receiving Flow Steering                                                                       |
| RGS        | Receiving Flow Steering                                                                                 |
| SBL        | Secondary Boot Loader                                                                                   |
| TLMM       | Top-Level Mode Multiplexer                                                                              |
| PLCSW      | PLC Host Software                                                                                       |
| LLDP       | Link Layer Discovery Protocol                                                                           |
| IEEE1905.1 | Standard to enable Hybrid devices—devices with multiple MAC/PHY interfaces: Wi-Fi, HomePlugAV, Ethernet |
| nVoy       | A certificate program for hybrid networking based on 1905 standard                                      |

### 19.24.1.2 Related documents

These documents provide specialized programming information for IPQ4018/IPQ4028.

| Doc number  | Title                                                            |
|-------------|------------------------------------------------------------------|
| 80-Y9347-18 | Preliminary IPQ4018 Access Point SoC Device Specification        |
| 80-Y9347-28 | Preliminary IPQ4028 Access Point SoC Device Specification        |
| 80-Y9348-28 | IPQ40x8 Device Revision Guide                                    |
| 80-Y6399-3  | IPQ40x8 QSDK Setup User Guide                                    |
| 80-Y9571-4  | IPQ40x8 Software User Guide                                      |
| 80-Y9571-5  | Lauterbach for IPQ40x8 Application Note                          |
| 80-Y9571-6  | IPQ40x8 Performance Management App. Note                         |
| 80-Y9571-7  | IPQ4018/IPQ4019/IPQ4028/IPQ4029 Home Switch SDK User Guide       |
| 80-Y9571-8  | IPQ4018/IPQ4019/IPQ4028/IPQ4029 Switch SDK Reference Manual      |
| 80-Y9571-9  | IPQ40x8/IPQ4019/IPQ4028/IPQ4029 SSDK Diagnostic Shell User Guide |
| 80-Y9571-10 | IPQ40x8 Switch UCI Command User Guide                            |
| 80-Y9572-1  | IPQ4019.ILQ.1.0 ES Firmware Build Guide                          |
| 80-Y9700-1  | IPQ4018/4028 AP.DK01 Hardware Reference Guide                    |

|            |                                        |
|------------|----------------------------------------|
| 80-Y9700-2 | IPQ4018/4028 AP.DK01 Setup Guide       |
| 80-Y9700-3 | IPQ40x8 RF Test User Guide (for QDART) |

## 19.24.2 Software architecture overview

The following figure shows the software architecture:

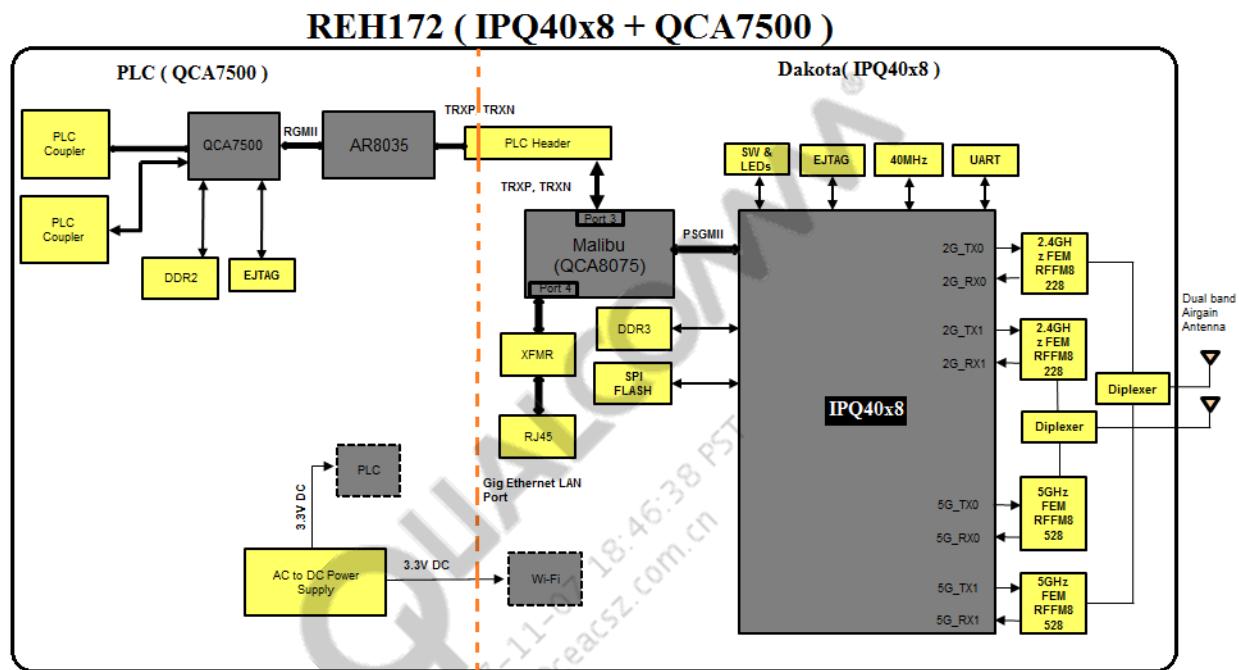


Figure 19-16 Software architecture

The following figure shows the IPQ40x8 hardware architecture:

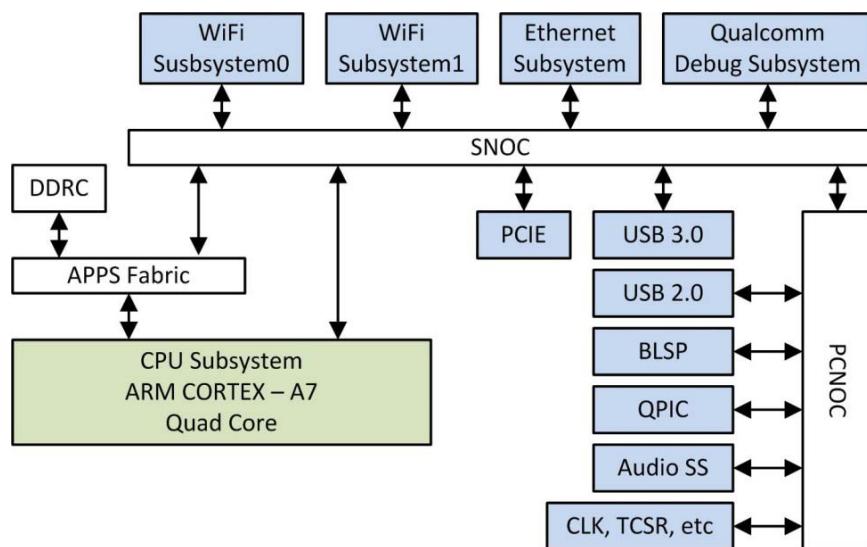


Figure 19-17 IPQ40x8 hardware architecture

### 19.24.2.1 CPUs

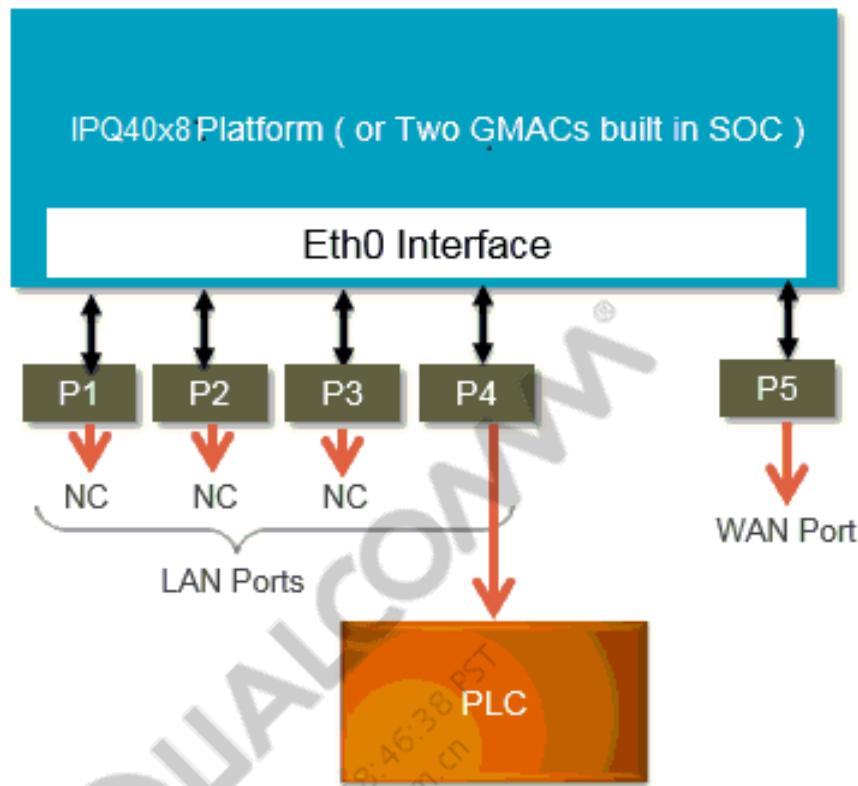
|                                       |                                                                                                                                                                                                                            |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Quad Cortex-A7 application processors | Qualcomm supplies a reference networking application platform consisting of Linux with OpenWrt. Configuration files tailor the networking software to suit the targeted applications. Core 0 is the first CPU to power up. |
| Dual Tensilica (WSS)                  | Responsible for Wi-Fi connectivity.                                                                                                                                                                                        |
| QCA7500                               | ARM11-Integrated HPAV2 MAC and PHY, integrated dual-channel line driver, supported HPAV configurations, 67 MHz or 30 MHz MIMO; 67 MHz or 30 MHz SISO; 67 MHz SISO + Smartlink                                              |

### 19.24.2.2 Reference software platform (QSDK)

- Linux kernel
- SBL
- Qualcomm Secure Execution Environment software
- U-boot: High-level boot loader for Linux
- OpenWrt: Embedded Linux distribution reference software and framework for customizing and building a networking application from existing and value-added components
- Board configuration files for the reference designs supplied by Qualcomm
- Application profiles that tailor the OpenWrt networking software to suit various application classes
- SoC drivers
- Qualcomm Wi-Fi proprietary driver package
- Qualcomm PLC host software
- Qualcomm PLC firmware and PIBs

### 19.24.3 REH172 in routing mode

- For REH172 routing mode support, IP packets must be routed between eth1 and eth0 interface with a different subnet programming for LAN and WAN traffic separation and with network address translation (NAT) applied.
- The default machine ID for REH172 is 8010007 so that eth0 single port group is formed for P4 and P5 if destination MAC is learned within S17 switch ports (namely, P4 and P5). Thereafter, traffic is forwarded without L3 packet inspection by host.
- For routing, traffic must be transmitted through the host or CPU for Layer 3 routing support.



**Figure 19-18 DK05 Default Port association in each group**

P1, P2, and P3 Malibu physical ports are not connected.

#### 19.24.3.1 Configure REH172 routing mode

Port reassociation procedure at EDMA and corresponding switch / SSDK configuration are also required. This configuration is similar to that for DK01, that is, Port 5 in group1, and Port 1, Port 2, Port 3, and Port 4 in group2.

1. Run the following commands for port reassociation using sysctl bit mapping:

```
cd /proc/sys/net/edma/
echo 0x20 > default_group1_bmp
echo 0x1E > default_group2_bmp
echo 2 > default_group1_vlan_tag
echo 1 > default_group2_vlan_tag
```

Customers or original device manufacturers (ODMs) can add the preceding commands in their startup script.

2. Edit the network configuration as follows. The underlined options are expected to be modified, based on the network configuration environment. The IP addresses of WAN and LAN, and ‘proto’ options can be modified by customers/ODMs.

```
config interface 'lan'
option ifname 'eth1'
```

```

        option force_link '1'
        option type 'bridge'
        option proto 'static'
        option netmask '255.255.255.0'
        option ip6assign '60'
        option ieee1905managed '1'
        option ipaddr '192.168.1.9'

    config interface 'wan'
        option ifname 'eth0'
        option proto 'static'
        option ipaddr '192.168.1.9'
        option netmask '255.255.255.0'

    config switch
        option name 'switch0'
        option reset '1'
        option enable_vlan '1'

    config switch_vlan
        option device 'switch0'
        option vlan '1'
        option ports '0t 1 2 3 4'

    config switch_vlan
        option device 'switch0'
        option vlan '2'
        option ports '0t 5'

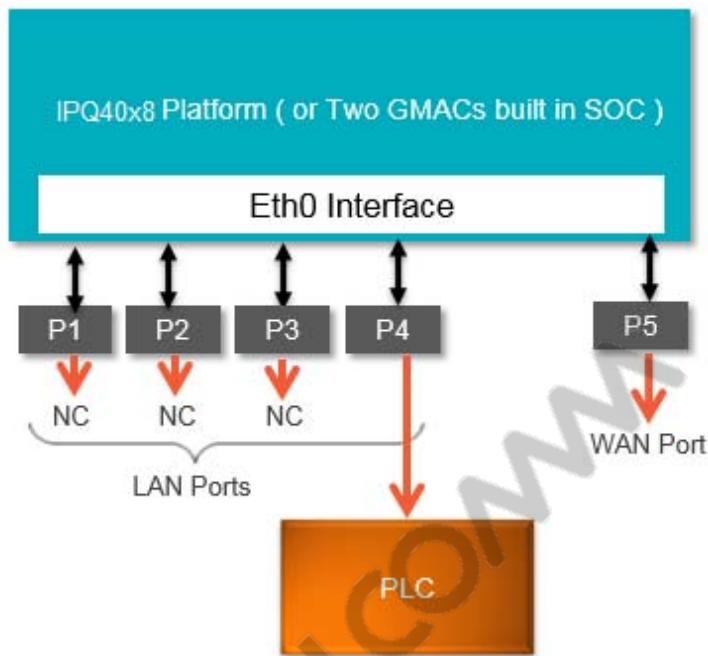
```

3. Perform a network restart.

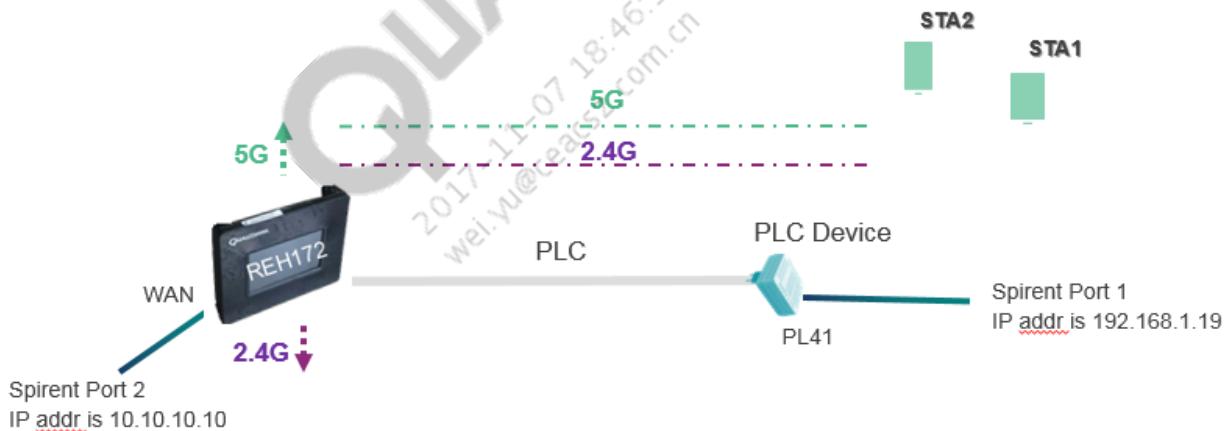
```
/etc/init.d/network restart
```

### 19.24.3.2 Connect REH172 routing topology

- After port remapping, two Ethernet interfaces are created, namely, eth1 for LAN and eth0 for WAN interface.
- PLC QCA7500/QCA7550 are attached to P4.
- Inbound traffic on P4, P5 is first forwarded to CPU/Linux L2 bridge if two subnet IP addresses are programmed on eth1 and eth0 interfaces, and then traffic redirected to Layer 3.



The following diagram illustrates bidirectional traffic between port 1 and port 2:



#### On REH172 :

- Br-lan is 192.168.1.9
- eth1 attached to br-lan
- Eth0 is 10.10.10.1

### 19.24.3.3 Example: Traffic stream creation for routing and NAT

#### WAN to LAN direction

- Source IP address—10.10.10.10
- Destination IP address—10.10.10.1
- Gateway IP address—10.10.10.1

## LAN to WAN direction

- Source IP address—192.168.1.19
- Destination IP address—10.10.10.10
- Gateway IP address —192.168.1.9

## Hardware NATing programming

Enable HNAT using the shell command:

```
ssdk_sh nat global set enable disable
iptables -t nat -A POSTROUTING -p udp -s 192.168.1.19 -j SNAT --to
10.10.10.1
iptables -t nat -I PREROUTING -p udp -i eth0 -s 10.10.10.10 -d 10.10.10.1
-j DNAT --to 192.168.1.19
```

In total, IPQ40x8 supports acceleration up to 1024 flows. Only one stream is created in this example.

## HNAT enable UCI configuration

|                     |                                                                                                                                   |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Enable HNAT mode 0: | <pre>config switch_ext   option device 'switch0'   option name 'NatGlobal'   option status 'enable'   option sync 'disable'</pre> |
| Enable HNAT mode 1: | <pre>config switch_ext   option device 'switch0'   option name 'NatGlobal'   option status 'enable'   option sync 'enable'</pre>  |

## HNAT disable UCI configuration

To disable HNAT, enter the following shell command:

```
ssdk_sh nat global set disable disable
```

|               |                                                                                                                                    |
|---------------|------------------------------------------------------------------------------------------------------------------------------------|
| Disable HNAT: | <pre>config switch_ext   option device 'switch0'   option name 'NatGlobal'   option status 'disable'   option sync 'disable'</pre> |
|---------------|------------------------------------------------------------------------------------------------------------------------------------|

## HNAT debug

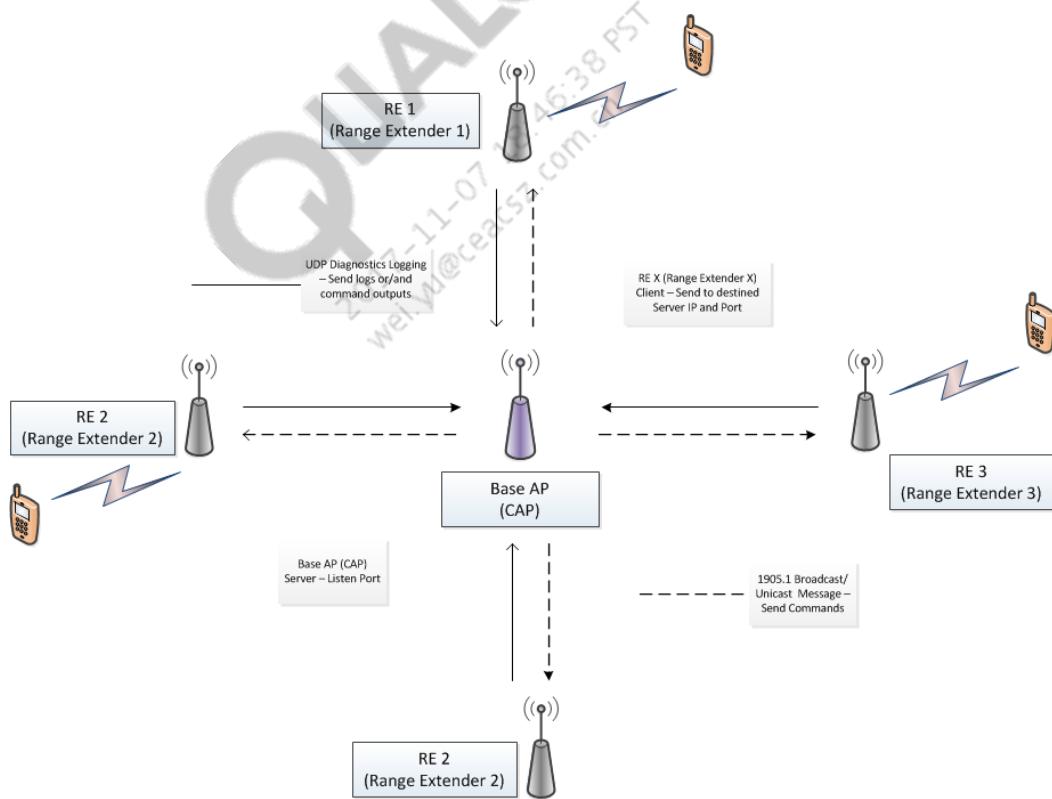
- To display NAT entry, enter `ssdk_sh nat natentry show`.
- To display NAPT entry, enter `ssdk_sh nat naptentry show`.

## 19.25 Wi-Fi SON: Diagnostics and troubleshooting

This section describes the Wi-Fi SON diagnostics mechanism for debugging and logging of messages to remote server AP. In multi-AP scenarios where CAP and REs are placed at varying distances, it is difficult to debug and enable logs dynamically in different APs. Wi-Fi SON diagnostics methodology allows user to log messages from different APs of the same network to a server (CAP). This Wi-Fi SON diagnostics capability also enables user to debug dynamically by use of ht commands. This section explains the design rationale and the configuration of the diagnostics mechanism for debugging and logging operations.

### 19.25.1 Wi-Fi SON diagnostics design

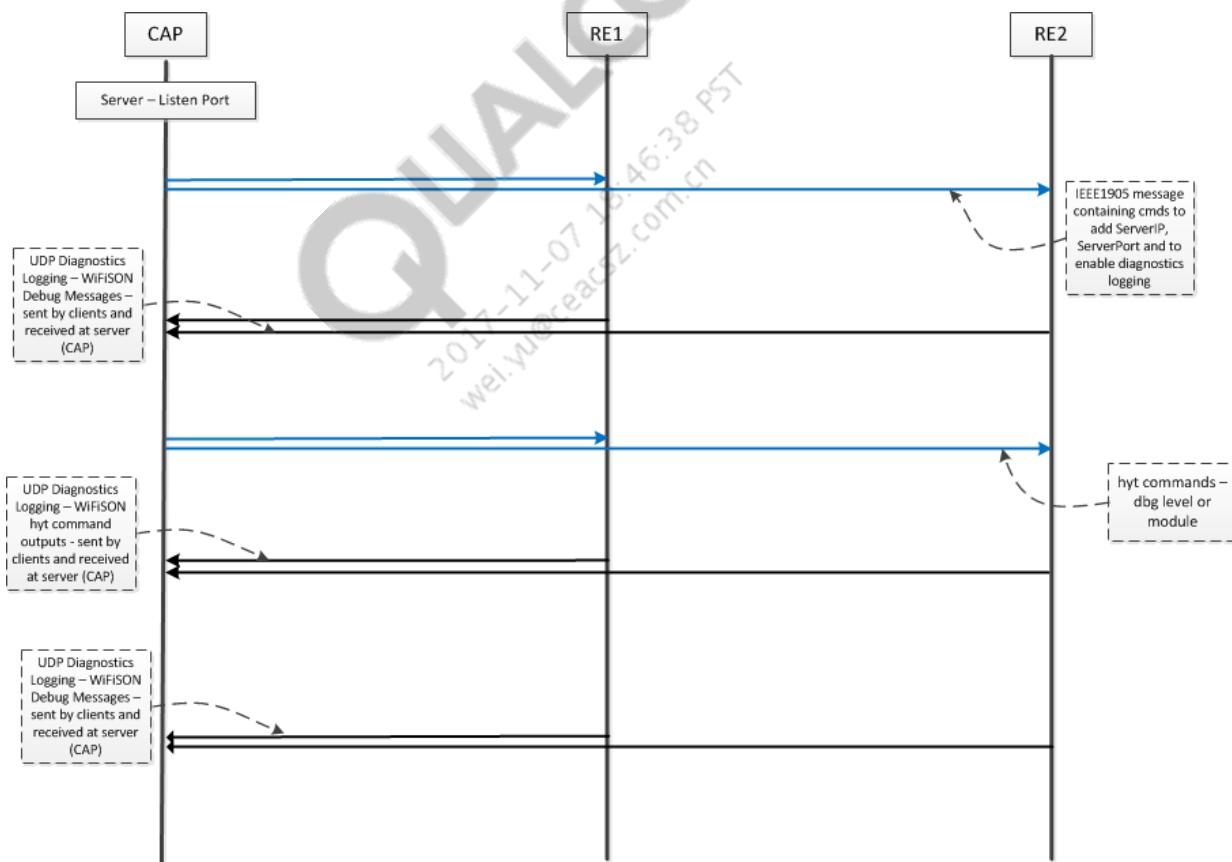
CAP acts a listening server on a destined port. CAP issues Diagnostics Configuration commands to a particular AP or to all APs in the network. CAP also issues ht commands. RE receives the configuration commands and programs them locally. Once Diagnostics logging is enabled, REs send the logs prepended with local MAC address to CAP via UDP stream. And in a similar manner, if ht commands are requested from CAP, REs send ht command outputs to CAP.



### 19.25.2 Wi-Fi SON diagnostics algorithm

The following sequence of events occur with the Wi-Fi SON diagnostics algorithm:

1. Create a listening port on CAP being server.
2. CAP unicast or broadcast diagnostic configuration commands through IEEE1905 message.
3. Repeater AP receives commands from CAP through IEEE1905 message.
4. Repeater AP parses the received commands and configures diagnostic logging module.
5. Repeater AP appends local MAC address to log messages and sends the same to the Server through UDP stream.
6. CAP receives through the listening port and prints to console.
7. CAP unicast or broadcast hyt commands through IEEE1905 message.
8. Repeater AP receives commands from CAP through IEEE1905 message.
9. Repeater AP parses the received commands and sends the output of the command to console and to server through UDP stream (if diagnostic logging is enabled).
10. CAP receives through the listening port and prints to console.



### 19.25.3 Functionality and components

The following are the Wi-Fi SON diagnostic components:

- *diag\_setServerIP*—Program Server IP address
- *diag\_setServerPort*—Program Server Port

- *diag\_enableLog*—Enable diagnostics logging
- *diag\_parseCmd*—Parse commands received from server (CAP).
- *diagOpenSocket*—Open the socket for the destined server IP and Port that will be used to send diagnostic logging records.
- *diagCloseSocket*—Close the socket that was being used for diagnostic logging.
- *diag\_write*—Write an arbitrary buffer into the log message currently being constructed.
- *diag\_finishEntry*—Complete the in progress log entry, sending it over the network to server (CAP).
- *listenOpenSocket*—Open the socket that will be used to receive diagnostic logging records.
- *listenToPort*—Listen to the port receiving diagnostic logging records and displaying it to console or stdout.

#### 19.25.4 Configure Wi-Fi SON diagnostics

After the image is flashed and configured, run the following commands on the CAP to enable diagnostic logging:

```
# qca_event_sample sendcmd xx:xx:xx:xx:xx:xx "DiagServerIP <CAP IP Address>"  
# qca_event_sample sendcmd xx:xx:xx:xx:xx:xx "DiagServerPort <Port Number>"  
# qca_event_sample sendcmd xx:xx:xx:xx:xx:xx "DiagEnable <1/0>"  
# qca_listen_port -P <Port Number>
```

Also, hyt commands can be sent to repeater(s) as follows:

```
# qca_event_sample sendcmd xx:xx:xx:xx:xx:xx "<hyt commands>"
```

An example to enable diagnostic logging is as follows:

```
# qca_event_sample sendcmd 00:03:7f: ef:d3:64 "DiagServerIP 192.168.1.2"  
# qca_event_sample sendcmd 00:03:7f: ef:d3:64 "DiagServerPort 5555"  
# qca_event_sample sendcmd 00:03:7f: ef:d3:64 "DiagEnable 1"  
# qca_listen_port -P 5555  
  
# qca_event_sample sendcmd 00:03:7f: ef:d3:64 "dbg level stamon dump"  
# qca_event_sample sendcmd 00:03:7f: ef:d3:64 "stadb s"
```

An example to enable diagnostic logging for broadcast is as follows:

```
# qca_event_sample sendcmd 01:00:00:00:00:00 "stadb s"
```

# 20 Coexistence of WLAN with NoDS mesh, BT, ZigBee, and CSRmesh

---

This chapter discusses the NoDistribution System (NoDS) frame support for mesh networks. Also, this chapter describes the coexistence of WLAN with Bluetooth (BT), ZigBee protocols, and CSRmesh devices. ZigBee is a suite of high level communication protocols used for personal area networks (PANs). ZigBee uses the IEEE 802.15.4 Wireless Personal Area Networks Standard and operates at 250 Kbps in 2.4 GHz band and at 40 Kbps or 20 Kbps in 900 / 868 MHz bands. Reduced cost, optimized power and long ranges are significant benefits of ZigBee. When the firmware receives beacon/management frame from the host, it sends a trigger to the COEX module.

## 20.1 NoDS Frame support for Mesh feature

Currently, the 10.4 WLAN driver does not provide interface to support Mesh networks. To enable customer to implement their own Mesh networks, enhancing WLAN driver for NoDS frame support. This document captures the detailed requirements for the NoDS frame support and high level design details.

### 20.1.1 Requirements

Following is the list of requirements covered as part of this feature support

- Separate Mesh VAP for mesh Tx/Rx packets
- Raw/Ethernet packet Transmit and Receive on Mesh VAP.
- Support to add Fake peer and corresponding parameters
- Support to add key entry for fake peer for data encryption/decryption for fake peer.
- Per Packet rate/retries transmission support for all data traffic sent via Mesh VAP interface.
- Per Packet Rx information for mesh packets.
- Per Packet Tx completion handler stats
- Disable disassociation frame/notification for mesh
- Rx filters for mesh
- Rx monitor filters in monitor mode

## 20.1.2 High Level Design

This section captures the interface details between Network stack to Host WLAN driver and Host to Firmware interface to support this feature. The interfaces are broadly classified in to two categories, control and Data.

- Control interface include the interface details for
  - Creating a Mesh VAP
  - Adding a Peer Mesh node
  - Inserting the Key to the added peer
- Data interface include the interface details for
  - Transmit interface to transmit raw data with custom meta header
  - Call back interface to provide feedback on the Transmit completions.
  - Raw packet receive interface.

### 20.1.2.1 Host Driver to Network Stack interface

#### Control Interface

Mesh support must be explicitly enabled with the following UCI command

```
uci set wireless.qcawifi.enable_mesh_support=1.
```

Control interface to the network stack is provided through the ioctl interface. Below are list of ioctl interface provide to the stack to support this requirement:

#### Interface to add a static peer

```
Ioctl: ieee80211_ioctl_setparam,          /* SIOCWFIRSTPRIV+0 */
Sub ioctl: IEEE80211_PARAM_ADD_LOCAL_PEER = 393
Number of arguments: 2 uint32 values
Arg1: <16 bit peerflags><byte5><byte4> of the Peer Mac address
Arg2: <byte3><byte2><byte1><byte0> of the Peer Mac address.
```

Each bit in peer flags indicate below capability.

|                           |                       |
|---------------------------|-----------------------|
| Dual Stream support       | BIT(0) i.e 0x1        |
| Three Stream support      | BIT(1) i.e 0x02       |
| Four Stream support       | BIT(2) i.e 0x04       |
| Reserved bits             | BIT(3) to BIT(7) 0xF8 |
|                           |                       |
| Peer support HT20         | BIT(8) i.e 0x0100     |
| Peer support HT20 & HT40  | BIT(9) i.e 0x0200     |
| Peer support 11ACVHT20    | BIT(10) i.e 0x0400    |
| Peer support 11ACVHT40    | BIT(11) i.e 0x0800    |
| Peer support 11ACVHT80    | BIT(12) i.e 0x1000    |
| Peer support 11ACVHT80_80 | BIT(13) i.e 0x2000    |
| Peer support 11ACVHT160   | BIT(14) i.e 0x4000    |
| Reserved bit              | BIT(15) ie. 0x8000    |

#### Interface to delete static peer

```
Ioctl: ieee80211_ioctl_kickmac,          /* SIOCWFIRSTPRIV+15 */
```

Number of arguments: character array (mac address)

Argument: <byte5>:<byte4>:<byte3>:<byte2>:<byte1>:<byte0> of the Peer Mac address.

### Interface to set mcast/ucast key

```
Ioctl: ieee80211_ioctl_setparam,      /* SIOCWFIRSTPRIV+2 */
Argument: character array (struct ieee80211req_key)
struct ieee80211req_key {
    u_int8_t    ik_type;      /* key/cipher type */
    u_int8_t    ik_pad;
    u_int16_t   ik_keyix;    /* key index */
    u_int8_t    ik_keylen;   /* key length in bytes */
    u_int8_t    ik_flags;
    u_int8_t    ik_macaddr[IEEE80211_ADDR_LEN];
    u_int64_t   ik_keyrsc;   /* key receive sequence counter */
    u_int64_t   ik_keytsc;   /* key transmit sequence counter */
    u_int8_t    ik_keydata[IEEE80211_KEYBUF_SIZE+IEEE80211_MICBUF_SIZE];
};

IEEE80211_ADDR_LEN is 6
IEEE80211_KEYBUF_SIZE is 32
IEEE80211_MICBUF_SIZE is 16
```

**NOTE** Take care of the endianness of the system while feeding input to set mcast/ucats keys through iwpriv, as the input being provided is in bytes. Provide input in the right order for ik\_keyix as it is declared to be a 16-byte variable and endianness will come into play.

### Interface to authorize static peer by the driver

```
Ioctl: ieee80211_ioctl_setparam,      /* SIOCWFIRSTPRIV+0 */
Sub ioctl: IEEE80211_PARAM_ALLOW_DATA = 395
Number of arguments: 2 uint32 values
Arg1: <byte5><byte4> of the Peer Mac address
Arg2: <byte3><byte2><byte1><byte0> of the Peer Mac address
```

**NOTE** The value of SIOCWFIRSTPRIV is 0x8BE0

## Data Transmit Interface

The following meta header is expected from network stack:

```
struct click_wifi_extra {
    uint8_t magic;    /* updated by Network Stack
    uint8_t flags;    /* updated by Network Stack
    uint8_t channel;  /* updated by WLAN driver (after TX completion)
    uint8_t keyix;    /* Updated by Network Stack
    uint8_t rssi;     /* updated WLAN driver (after TX completion)
    uint8_t silence;  /* updated by Network Stack
    uint8_t power;    /* Updated by Network Stack
    uint8_t retries;  /* Updated by Network Stack

    uint8_t max_tries[CLICK_WIFI_SCHEDULE_SIZE]; /* Updated by Network
    Stack
    uint8_t rates[CLICK_WIFI_SCHEDULE_SIZE]; /* Updated by Network Stack
```

```

        uint8_t unused[4]; /* None
    };
```

CLICK\_WIFI\_SCHEDULE\_SIZE is initialized to 2 in the driver (that is, 16 bytes of Meta header should be sent by network stack).

|    | <b>Field</b> | <b>Filled by</b> | <b>Used by</b> | <b>Information expected</b>                                                                                                                    |
|----|--------------|------------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| 1  | Magic        | Network Stack    | None           |                                                                                                                                                |
| 2  | Flags        | Network Stack    | Host           | BIT 6 i.e flag (1<<6) METAHDR_FLAG_NOENCRYPT -> Frame is not encrypted when this flag is set.                                                  |
| 3  | Channel      | Host             | Network Stack  |                                                                                                                                                |
| 4  | Keyix        | Network Stack    | Host           | This should be valid key ix value.                                                                                                             |
| 5  | Rssi         | Host             | Network Stack  |                                                                                                                                                |
| 6  | Silence      | Network Stack    | None           |                                                                                                                                                |
| 7  | Power        | Network Stack    | Host           |                                                                                                                                                |
| 8  | Retries      | Network Stack    | Host           | For software retries                                                                                                                           |
| 9  | Max_tries    | Network Stack    | None           | For QCA9980 FW this is invalid                                                                                                                 |
| 10 | Rates        | Network Stack    | Host           | bits (3:0)mcs. 0 to 9 ,<br>bits (5:4)nss. 0 = 1nss, 1= 2 nss, 2 = 3 nss, 3 = 4 nss<br>bits (7:6)pream type. 0 = ofdm, 1 = cck, 2 = ht, 3 = vht |

In transmit path, host will receive the preceding meta header from Network Stack. Some of the fields are to be filled by host driver after TX completion. This meta header is expected at the starting of the skb. Data packets are followed by this meta header.

The host copies the required fields meta header to `htt_tx_msdu_desc_t` and sends to the firmware. After transmitting the packet, the firmware provides RSSI for the mesh VAPs packets in Tx completion message. The host parses this message and fills the required fields in meta header.

For the mesh VAP packets, after Tx completion of mesh VAP packets happens, the host, instead of freeing the skb, accesses the callback function, which is registered at initialization time by network stack. It is the responsibility of the callback to free this skb.

### Callback function

At initialization, a callback must be registered by network stack for Tx completion function. The format of the callback function is `void tx_complete (struct sk_buff *skb);`. The `external_tx_complete` variable is exported for others to register this callback.

### Data Receive Interface

A packet that is received on the mesh VAP interface is directly sent to the network stack through `netif_rx` without any modification in the frame headers. The following per-packet information is updated in `skb->cb`.

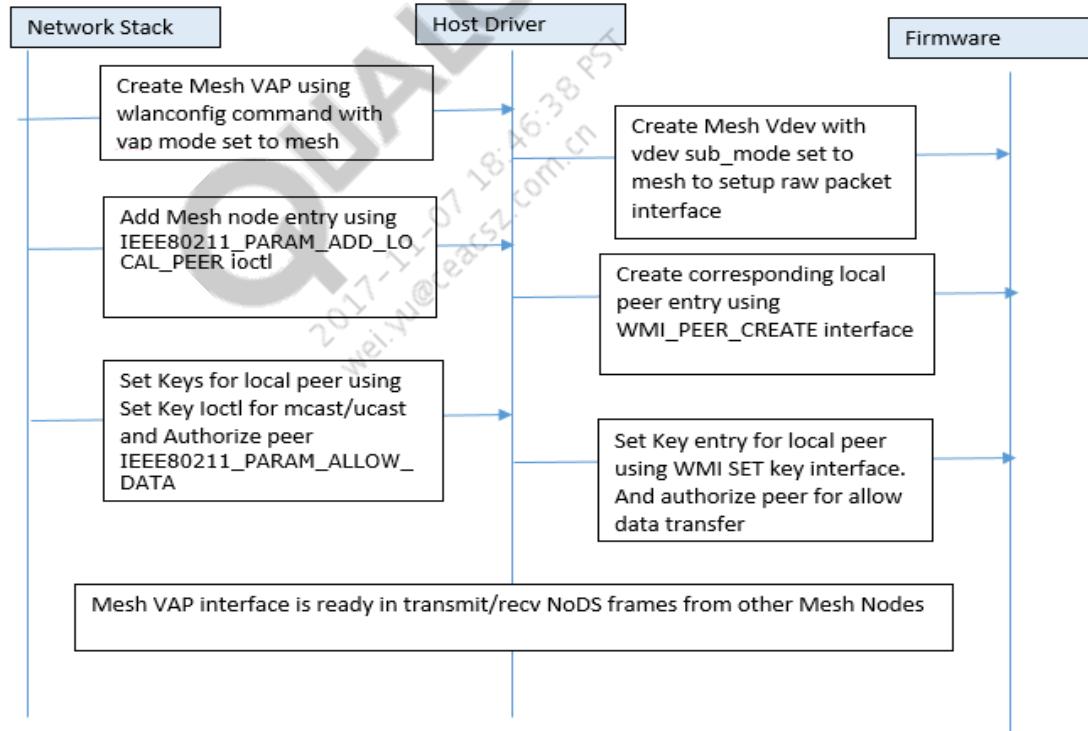
```
typedef struct per_msdu_mesh_data_recv_status {
```

```

        int8_t          rs_flags;
#define IEEE80211_RX_FIRST_MSDU      0x01
#define IEEE80211_RX_LAST_MSDU       0x02
/* first_msdu=1, last_msdu=1 ==> non-Aggregation frame*/
/* first_msdu=1, last_msdu=0 ==> first msdu*/
/* first_msdu=0, last_msdu=0 ==> msdu frame between first and last
msdu*/
/* first_msdu=0, last_msdu=1 ==> last msdu*/
    int16_t          rs_rssi;           /* RSSI (noise floor adjusted) */
    u_int8_t         rs_channel;
    int              rs_ratephy1;
    int              rs_ratephy2;
    int              rs_ratephy3;
    u_int8_t         rs_decryptkey[32];
} per_msdu_mesh_data_rx_status;

```

### 20.1.2.2 Sequence Diagram for setting up Mesh Interface



## Frame Formats

### Transmit Path

Frame format should be as per Wi-Fi standard without FCS field and should be after Meta header which is 16 bytes. Meta header format is as mentioned 3.1.2 section. Length of the fields will change based on the different parameters and should be as per standard. Following are examples

### Non Security Frames

|             |                   |            |      |
|-------------|-------------------|------------|------|
| Meta Header | 802.11 MAC Header | LLC Header | Data |
|-------------|-------------------|------------|------|

### Security Frame - WPA2 PSK

|             |                   |    |            |      |          |
|-------------|-------------------|----|------------|------|----------|
| Meta Header | 802.11 MAC Header | IV | LLC Header | Data | MIC Tail |
|-------------|-------------------|----|------------|------|----------|

IV, MIC are 8 bytes each.

**NOTE** In general raw mode frames should have place holders for filling IV, MIC, etc. Depending on the security mode, CRC will be added by hardware and place holder is not required.

### Receive Path

Frames will be same as seen on air including FCS.

## 20.1.3 Additional feature list

- Enable Beacons on Mesh VAP
- Mesh local peer timeout
- Mesh local peer cap re-intersect

### 20.1.3.1 Enable Beacons on Mesh VAP

Currently when Mesh VAP is created, it does not send out Beacon by default. Purpose of this feature is to provide user control to enable/disable Beacons on Mesh VAP.

#### Design and Implementation

Host driver needs to explicitly send WMI\_VDEV\_PARAM\_CAPABILITIES with WMI\_VDEV\_BEACON\_SUPPORT set and WMI\_VDEV\_WDS\_LRN\_ENABLED unset to enable Beacon on Mesh VAP.

Changes that were done to enable this feature:

- The “meshcap” and “get\_meshcap” ioctls are introduced to provide capabilities to Mesh VAP, for example, enable Beacon
  - First bit in meshcap defines enable/disable Beacon
- New parameter iv\_mesh\_cap in VAP structure to store meshcap value provided by user
- Check in ol\_ath\_vap\_up(), if it is mesh vap and iv\_mesh\_cap has Beacon enabled, send FW WMI\_VDEV\_PARAM\_CAPABILITIES set as 1 to enable Beacons.

## Steps to Enable Beacon on Mesh VAP

- In etc/config/wireless, add the following setting:  

```
option mode mesh
```
- Enable mesh support using the following command:  

```
uci set wireless.qcawifi=qcawifi
uci set wireless.qcawifi.enable_mesh_support=1
uci commit
```
- Bring up wifi driver using “wifi” command
- When Mesh interface/vap is up (for example, ath0), use following ioctl to enable Beacons  

```
iwpriv ath0 meshcap 0x1
```
- Bring Mesh interface down and up to take effect of previous ioctl  

```
ifconfig ath0 down
ifconfig ath0 up
```
- Beacons will now be seen from Mesh VAP

### 20.1.3.2 Mesh local peer timeout

In Mesh mode, Mesh VAP will be continuously receiving beacons from all mesh peers in range, after getting the beacons, if it's a new mesh peer, then it will be added locally to mesh VAPs node table.

If no beacon received from a specific mesh peer beyond some time, the local mesh node should be timed out and deleted.

#### Design and Implementation

On AP, the scan module will continuously receive all beacons on the channel, and save and update all necessary information needed for this feature, we'll just use the scan module. Also there's a per-radio inactivity timer running every 1 second.

So the design is to:

- Use the inactivity timer, for every 60 seconds, check the timestamps of all the mesh peer scan entries in the scan table.
- If timestamp's after some threshold for example, 120 seconds, send an event to user space.
- User space application receives the events and decide what to do, usually user app will call STA kickout to delete the stations.

To reduce unnecessary operations, two mesh flags are added in node struct and scan entry struct.

In node struct, new flag for mesh added in ni\_ext\_flags: IEEE80211\_LOCAL\_MESH\_PEER.

In scan table's scan entry struct, flag se\_flag added, and flag for mesh (IEEE80211\_SE\_FLAG\_IS\_MESH) is also added.

At run time, ic\_inact\_timer handler function ol\_ath\_timeout() will be ran every 1s. And every 60s, it will call ieee80211\_check\_timeout\_mesh\_peer() to check mesh peers for timeout. Only if

beacon from a mesh peer's received, the scan entry's timestamp will be updated. So no beacon received in some time, the scan entry will be timed out.

### Steps to test Mesh local peer timeout

- Set up Mesh VAP on 2 APs
- Enable beacon on both sides
- Enable debug level  

```
iwpriv ath0 dbgLVL 0x8
iwpriv ath0 dbgLVL_high 0x800
```
- Scan entry timeout to 5 seconds  

```
iwpriv ath0 scanentryage 5
```
- Add local peers on both APs
- Enable acfg event capture  

```
acfg_tool -e &
```
- Disable beacon on one AP  

```
ifconfig ath0 down
```

The following prints are shown on another AP console:

```
ieee80211_check_timeout_mesh_peer: [0x8c:0xfd:0xf0:0x01:0x05:0x1f] mesh
peer timed out.
root@OpenWrt:~# cat /etc/acfg_event_log
ath0: Session timeout for STA 8c:fd:f0:01:05:1f
```

### 20.1.3.3 Mesh local peer cap reintersect

In mesh mode, Mesh VAP will be continuously receiving beacons from all mesh peers in range, after getting the beacons. If any capabilities (mentioned below) changed, the capabilities of the corresponding local mesh peer node should be updated same.

- HT20/HT40/VHT
- Supported Rates
- (AMSDU/AMPDU) aggregation on/off

### Design and Implementation

On AP, the scan module will continuously receive all beacons on the channel, and save and update all necessary information needed for this feature, we'll just use the scan module.

So the design is to:

- In the scan module, when updating scan entries after receiving beacons, check for mesh vap and mesh local peer (node)
- If any capabilities changed in beacon, redo the capability intersection between the node and the mesh VAP.

A new function ieee80211\_beacon\_intersect() was added to do intersection using beacons, referred to AP codes' ieee80211\_recv\_asreq().

To detect changes in the beacon, a checksum of beacon's interested part will be calculated for every beacon and saved in corresponding scan entries. If checksum does not match, then redo intersection.

A new umac module parameter called 'enable\_mesh\_peer\_cap\_update' was added for this feature, to enable it, do: insmod umac.ko enable\_mesh\_peer\_cap\_update=1.

### Steps to test Mesh local peer cap re-intersect

- Setup and bring up mesh vaps on 2 APs

/etc/config/wireless example:

```
config wifi-device 'wifi0'
    option type 'qcawifi'
    option macaddr '8c:fa:f0:00:e2:11'
    option hwmode '11ac'
    option disabled '0'
    option channel '149'
config wifi-iface
    option network 'lan'
    option device 'wifi0'
    option ssid 'MESH_AP1'
    option mode 'mesh'
config qcawifi 'qcawifi'
    option nss_wifi_olcfg '0'
    option enable_mesh_support '1'
    option enable_mesh_peer_cap_update '1'
```

- Set proper debug level for prints

```
iwpriv ath0 dbgLVL 0x8
iwpriv ath0 dbgLVL_high 0x800
```

- Enable Mesh VAP beaconing on both APs

```
ifconfig ath0 down
iwpriv ath0 meshcap 0x1
ifconfig ath0 up
```

- Add local peers on both APs

- MESH AP1

```
iwpriv ath0 addlocalpeer 0x12048CFD 0xF001051F
```

- MESH AP2

```
iwpriv ath0 addlocalpeer 0x12048CFD 0xF0013FDE
```

- Result for first time intersection

When adding local peer, if there's any beacon with same mac address on the air as the peer we added, the first time intersection will be done between the received beacon and the Mesh VAP. AP console will show 'first time mesh peer cap intersection...',

Example:

```
ieee80211_create_new_scan_entry: 8C:FD:F0:01:64:5D SSID=MESH1 chan= 36
p=d8c70600
ieee80211_scan_entry_update: [8c:fd:f0:01:64:5d] first time mesh peer cap
intersection ...
vap-0(ath0):legacy rates:
vap-0(ath0):0x2498128c
vap-0(ath0):0x6c6048b0
vap-0(ath0):0x0
vap-0(ath0):ht rates:
vap-0(ath0):0x3020100
vap-0(ath0):0x7060504
vap-0(ath0):0xb0a0908
vap-0(ath0):0xf0e0d0c
vap-0(ath0):0x13121110
vap-0(ath0):0x17161514
vap-0(ath0):0xb1a1918
vap-0(ath0):0x1f1e1d1c
vap-0(ath0):0x0
vap-0(ath0):ASSOC SUCCESS: MAC:8c:fd:f0:01:64:5d PHYMODE=0xa NSS=4
FLAGS=0x621b003 max_mpdu=1048575 HT numrates=32 maxratevht=0xff VHT: rx_
mcs_set=0xffaa tx_mcs_set=0xffaa
[wifi0] FWLOG: [142295] WAL_DBGID_SECURITY_ALLOW_DATA ( 0x446ae4 )
[wifi0] FWLOG: [142295] RATE: ChainMask f, peer_mac 64:5d, phymode 10,
ni_flags 0x0621b003, vht_mcs_set 0xffaa, ht_mcs_set 0xffffffff, legacy_
rate_set 0x0000
```

- Change parameters such as PHY mode/bandwidth on MESH2 on AP2

```
ifconfig ath0 down
iwpriv ath0 mode 11NAHT20
ifconfig ath0 up
```

- Result for beacon cap change on AP1

After AP2's cap changed in the beacon, on AP1's console, information regarding 'mesh peer cap changed' is displayed as follows:

```
ieee80211_scan_entry_update: [8c:fd:f0:01:05:1f] mesh peer cap changed,
do intersection...
vap-0(ath0):legacy rates:
vap-0(ath0):0x2498128c
vap-0(ath0):0x6c6048b0
vap-0(ath0):0x0
vap-0(ath0):ht rates:
vap-0(ath0):0x3020100
vap-0(ath0):0x7060504
vap-0(ath0):0xb0a0908
vap-0(ath0):0xf0e0d0c
vap-0(ath0):0x13121110
vap-0(ath0):0x17161514
vap-0(ath0):0xb1a1918
vap-0(ath0):0x1f1e1d1c
vap-0(ath0):0x0
```

```
vap-0(ath0):ASSOC SUCCESS: MAC:8c:fd:f0:01:05:1f PHYMODE=0x6 NSS=4
FLAGS=0x203003 max_mpdu=65535 HT numrates=32 maxratevht=0x0
[wifi0] FWLOG: [161593] WAL_DBGID_SECURITY_ALLOW_DATA ( 0x446ae4 )
[wifi0] FWLOG: [161593] RATE: ChainMask f, peer_mac 5:1f, phymode 6, ni_
flags 0x00203003, vht_mcs_set 0x0000, ht_mcs_set 0xffffffff, legacy_rate_
set 0x0000
```

## 20.2 Bluetooth WLAN coexistence

Bluetooth WLAN coexistence feature captures the BT/ZB coexistence software design for Dakota/QCA9984.

It provides a high level design for the implementation of BLUETOOTH/ZigBee coexistence from host and firmware perspective for Dakota/QCA9984.

The following details are discussed in this section:

- COEX algorithm
- HOST <--> Firmware interface
- User space <--> Host interface

### 20.2.1 Assumptions

#### **Assumption 1:**

WLAN has no control over BT/ZB high priority. BT/ZB high priority traffic overrides all WLAN traffic.

#### **Assumption 2**

WLAN Beacon/management frames and WMM\_AC\_VO packets have higher priority than BT low priority traffic. BT low priority traffic is blocked until all the high priority WLAN frames in the queue are transmitted or timed out.

#### **Assumption 3**

BT high priority traffic should not block the WLAN for longer duration (more than few milliseconds).

#### **Assumption 4**

Mesh frames are assumed to have TID classification and no additional processing is performed by the COEX module to identify the priority of the mesh frames.

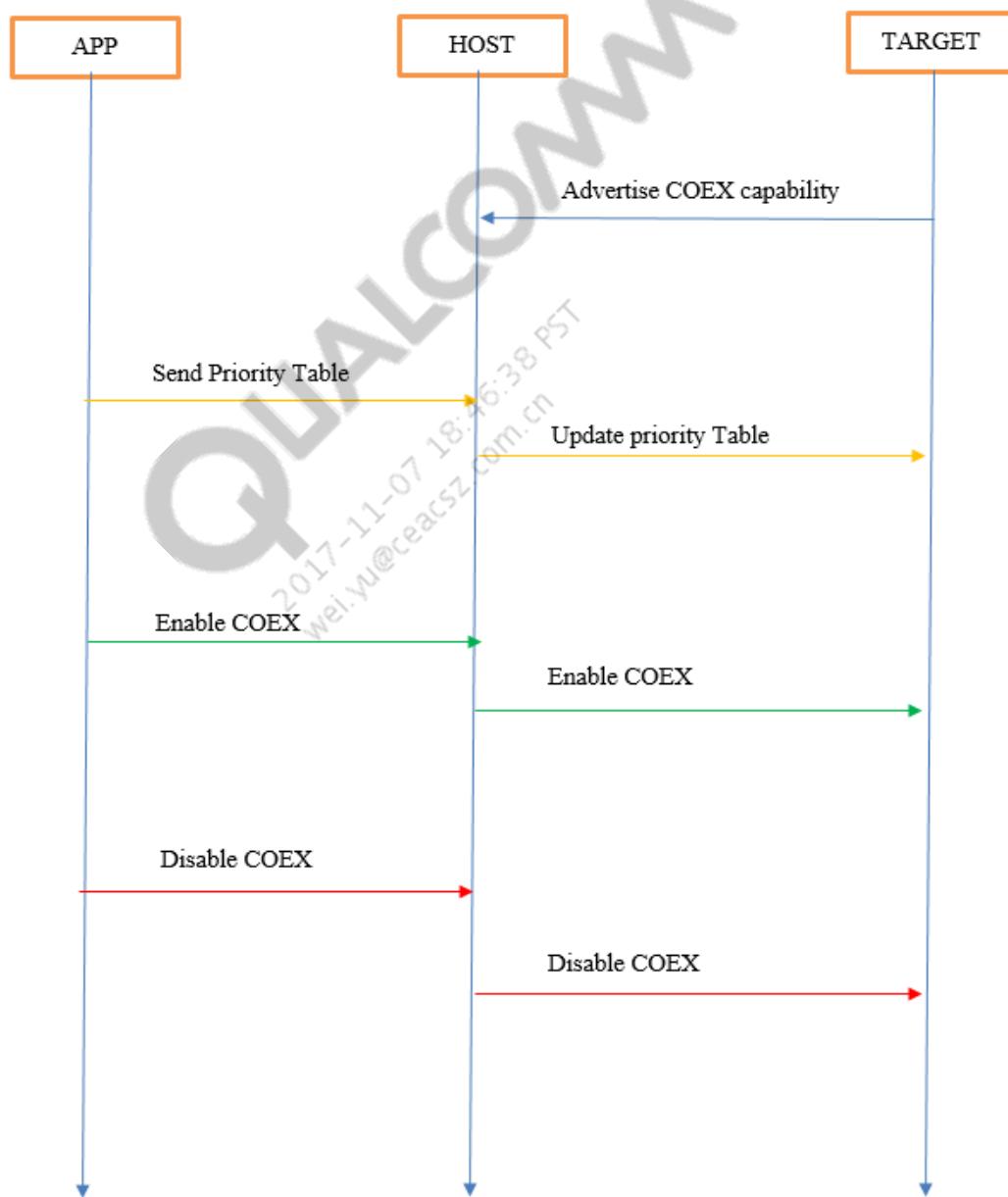
#### **Assumption 5**

BT low priority traffic is blocked irrespective of the beacon transmission type (staggered or burst) in multi VIF scenario.

### Assumption 6

It is assumed that BT traffic is not going to block the WLAN for long duration. Confirm if this assumption is true else logic to pause should be added to the design. TX processing since WLAN frames is retried by the MAC HW and may get dropped.

#### 20.2.2 COEX initialization



### 20.2.3 Priority table

**Table 20-1** shows the baseline priority table. This can be overridden from the host during initialization.

**Table 20-1 Priority table**

| Wi-Fi traffic class | Wi-Fi priority | WL_Priority_N pin<br>[0 – Block BT, 1 – Unblock BT] |
|---------------------|----------------|-----------------------------------------------------|
| WMM_AC_BE           | Low            | 1                                                   |
| WMM_AC_BK           | Low            | 1                                                   |
| WMM_AC_VI           | Low            | 1                                                   |
| WMM_AC_VO           | High           | 0                                                   |
| Management          | High           | 0                                                   |
| Beacon              | High           | 0                                                   |

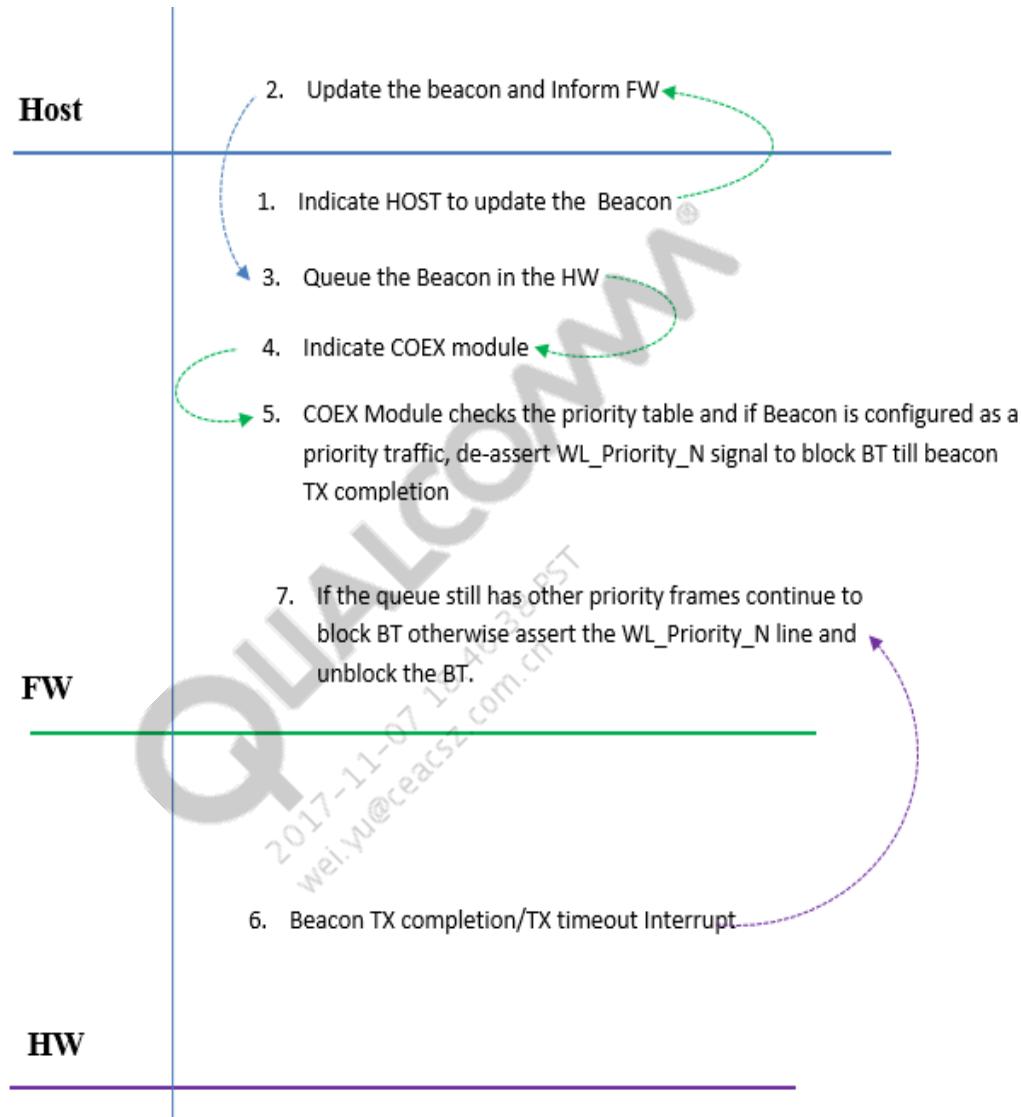
### 20.2.4 COEX module

When the firmware receives beacon/management frame from the host it sends a trigger to the COEX module.

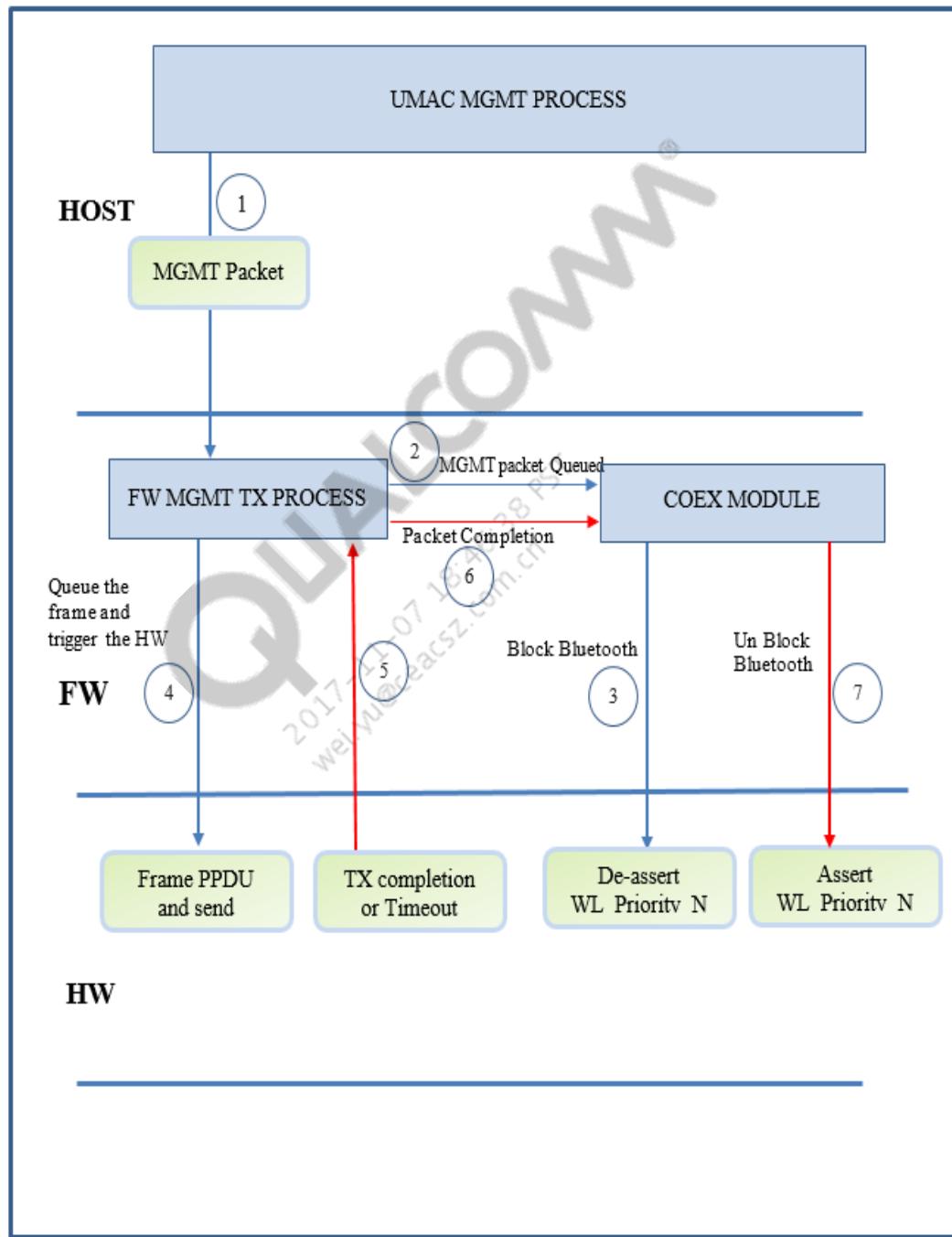
On receiving the trigger from the WLAN TX module, COEX checks if the WLAN frames in the queue (SW) are marked as priority frame in the priority table. If the frames are found to be a priority frame, COEX module will de-assert the WL\_Priority\_N signal to block the BT traffic. If the frames are marked as low priority then COEX module will assert the WL\_Priority\_N signal to allow the BT traffic.

Every Tx completion is notified to COEX module. On every Tx completion, the COEX module check if the queue still has priority WLAN traffic. If the queue contains priority traffic, WL\_Priority\_N signal is kept de-asserted until the queue becomes empty.

## 20.2.5 Beacon frame handling during COEX



## 20.2.6 Management Frame Handling during COEX



## 20.2.7 WMM\_AC\_VO Frame Handling during COEX

VO frames are handled in similar way as Beacon/management frames. COEX continues to block low priority BT traffic until VO frames are there in the queue. BT low priority is allowed only when the VO queue is empty.

## 20.2.8 HOST Driver Interface

Target advertises BT coex capability by setting WMI\_SERVICE\_BT\_COEX bit in service ready event.

Host checks this bit and btcoex support in device tree file and enables BT coex in resource config while sending WMI\_INIT\_CMD.

Host can set WLAN traffic priority using WMI\_BT\_COEX\_CFG\_CMDID by default Voice, Beacon and management frames have high priority.

Host can also dynamically disable or enable BT coex using

WMI\_PDEV\_PARAM\_ENABLE\_BT\_COEX.

Commands to set or get WLAN traffic priority and enable/disable BT coex feature

```
Set WLAN traffic priority.  
iwpriv wifi0 btcoex_wl_pri 0x34
```

Following are wlan priority bits and a bit set indicates particular WLAN traffic is of high priority.

Best effort - Bit 0

Background - Bit 1

Video - Bit 2

Voice - Bit 3

Beacon - Bit 4

Management- Bit 5

So 0x34 = 110100 implies video, beacon and management traffic are high priority.

To display current priority:

```
iwpriv wifi0 g_btcoex_wl_pri  
Enable BT coex feature  
iwpriv wifi0 btcoex_enable 1  
Disable BT coex feature  
iwpriv wifi0 btcoex_enable 0  
To display current status:  
iwpriv wifi0 g_btcoex_enable
```

By default video, management and beacon frames has high priority.

## 20.3 Coexistence of WLAN and Bluetooth for CSRMesh

Internet of Things (IoT) coexistence enables CSRmesh and Zigbee to operate on a WLAN AP. CSRmesh and Zigbee traffic can occur at random times. Data throughput is low, but low latency can be important to the user experience. Therefore, packet loss must be minimized. The ideal state for an IoT gateway is to be in receive mode 100% of the time to capture random transmissions from IOT devices. With the WLAN coexistence methodology that exists before the introduction of the coexistence functionality of WLAN with Bluetooth for CSRMesh, the AP can be in condition of not transmitting any data except for high-priority traffic. The AP behavior is enhanced enable WLAN traffic to continue to flow.

Specific time is allocated for BT and WLAN. For example, in a specific interval of 100 ms, 80 ms can be provisioned for WLAN and 20 ms can be provisioned for BT so that WLAN and BT low priority traffic are not affected. These time periods are configurable.

The design methodology of TDM (Time Division Mechanism) algorithm is enhanced to support coexistence between the colocated WLAN and Bluetooth on the WIN products. The TDM logic depends on the COEX design implemented on DK07/QCA9984. This functionality is applicable for the following IOT use cases:

- BT LE GATT
- CSR Mesh
- Zigbee

Keep the following points in mind for the WLAN and Bluetooth low energy (LE) coexistence:

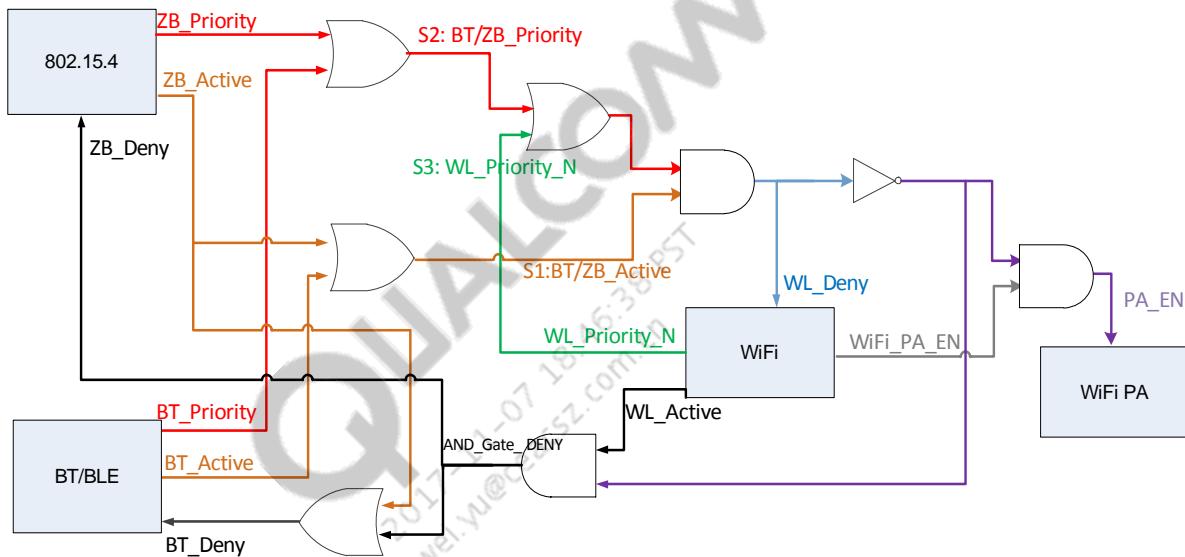
- If BT tx does not happen in BT duty cycle it will lead to BT connection issues.
- If all WLAN traffic is configured as low priority, it will be blocked during BT duty cycle leading to loss of beacons.
- Zigbee/BT traffic can cause rate drop of WLAN traffic and KPI are as measured.
- BT coex duty cycle will be support only in AP mode.
- COEX firmware module should provide a logic to configure the WLAN TX duty cycle.
- COEX firmware module should ensure that colocated WLAN TX is restricted only for the configured duty cycle.
- COEX firmware module should expose an interface through the host driver to configure the WLAN TX duty cycle
- During the BT period, the host can configure WLAN priority table to override BT low priority traffic. [This is to make sure Beacon frames are sent during the BT period]

### Hardware interfaces for IPQ4019

- BT active and BT priority GPIO from BT module and WLAN priority from WLAN module used to intimate that respective module is transmitting and used for co-existence to stop BT/WLAN so that other traffic and flow through without interference issues.
- Bluetooth module asserts BT active GPIO when it has to send BT traffic, it also asserts BT priority GPIO if the BT traffic is high priority.

- WLAN module de-asserts WLAN priority GPIO if there is high priority WLAN traffic and assert WLAN priority GPIO if WLAN traffic is low priority.
- BT priority and WLAN priority GPIOs are ORed and output is ANDed with BT active to generate LTE-ACTIVE which is fed as input to WLAN.
- If LTE-ACTIVE GPIO goes high, WLAN tx will be stopped and BT traffic will flow through.
- WLAN ACTIVE is ANDed with NOT of LTE-ACTIVE and fed as BT-DENY to BT module to stop BT traffic.

Above hardware logic ensures WLAN traffic is stopped if there is high priority BT priority and stops BT low priority traffic only when there is high priority WLAN traffic



- \* Allow BT high priority traffic irrespective of WLAN priority.
- \* Allow WLAN high priority traffic only when BT traffic priority is low.
- \* Allow BT low/high priority traffic when WLAN priority is low.

| P1:BT/ZB_ACTIVE | P2:BT/ZB_Priority | P3:WL_Priority_N | Priority             |
|-----------------|-------------------|------------------|----------------------|
| 0               | 0                 | 0                | WLAN TX goes through |
| 1               | 0                 | 1                | Stop WLAN TX         |
| 1               | 1                 | 1                | Stop WLAN TX         |
| 1               | 0                 | 0                | WLAN TX goes through |
| 1               | 1                 | 0                | Stop WLAN TX         |

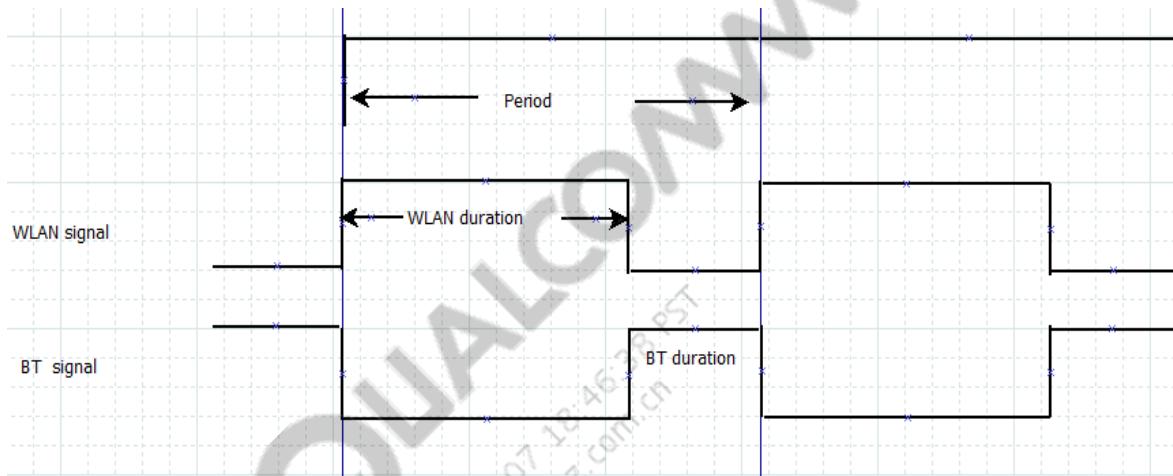
## Design overview

The time periods to be allocated for QLAN and BT traffic are configurable and behavior during WLAN and BT duty cycle is explained in this section. COEX firmware module takes two parameters—time for which WLAN TX is active and the period:

$$\text{WLAN TX Duty cycle} = \frac{\text{Time WLAN TX is active}}{\text{Period}}$$

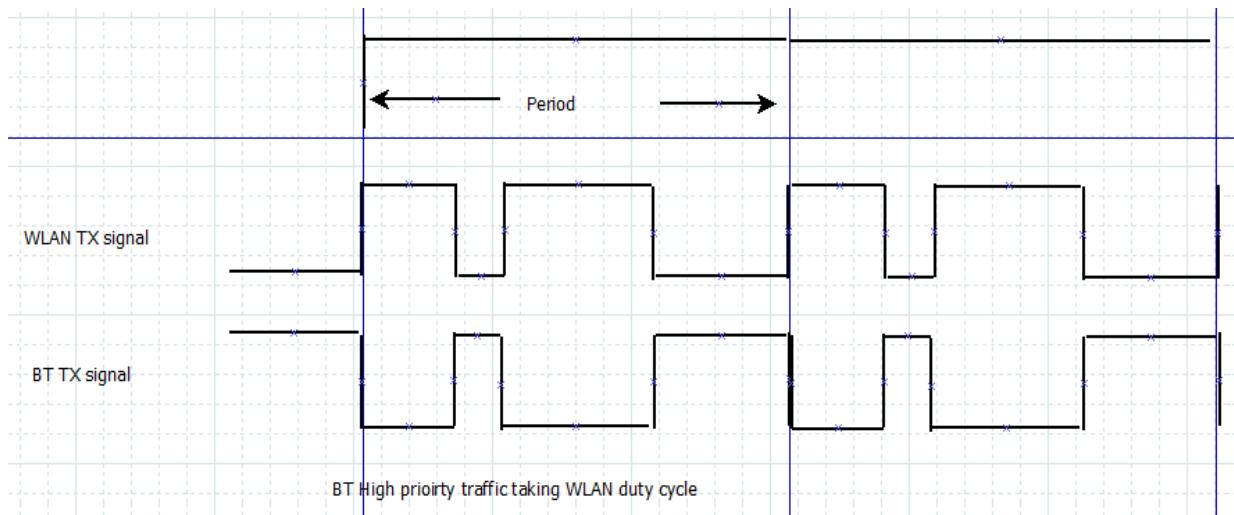
### Case 1:

BT does not interfere in WLAN duration and WLAN does not interfere in BT duration



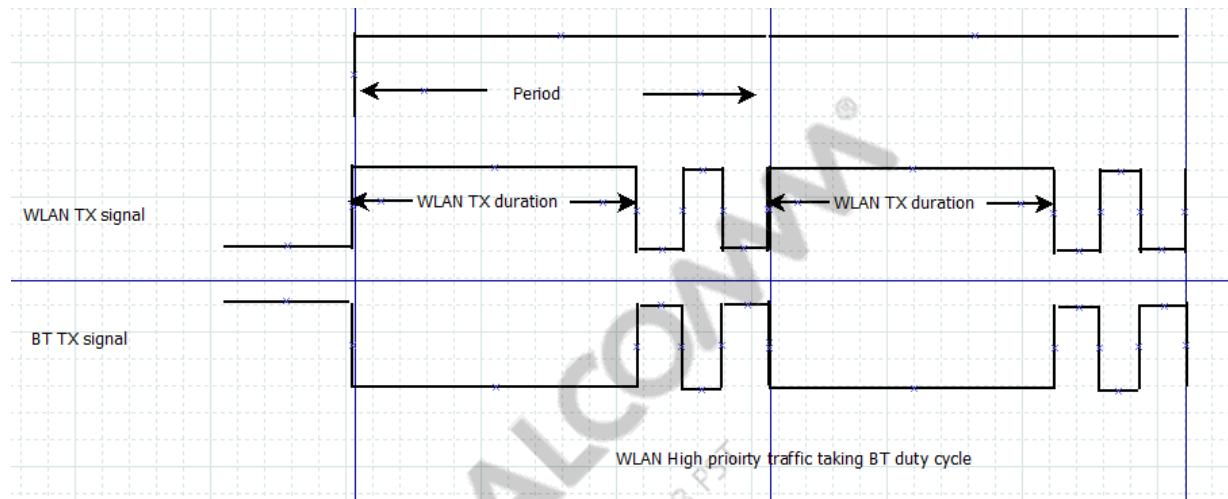
### Case 2:

BT high priority traffic overrides WLAN duration. For the configured duty cycle WLAN TX will be allowed and rest of the period WLAN TX queue is paused. In this case BT high traffic can override WLAN TX duty cycle.



### Case 3:

WLAN priority table overrides BT low priority traffic, WLAN frames are configured to have high priority over BT traffic. In this case only the queues which has low priority is paused and the frames in high priority queues are still allowed to be transmitted.



The baseline priority table for COEX with duty cycle is shown below. By default Voice, management and beacon WLAN traffic is of high priority and this WLAN priority can be configured from host.

### Configure BT coexistence

| Wi-Fi Traffic Class | BT Priority | Duty cycle | [0 – Block BT, 1 – Unblock BT] | WLAN_PRIORITY_GPIO |
|---------------------|-------------|------------|--------------------------------|--------------------|
| LOW                 | HIGH        | WLAN       | 1                              | 1                  |
| HIGH                | HIGH        | BT         | 1                              | 0                  |
| HIGH                | LOW         | WLAN       | 0                              | 0                  |
| LOW                 | LOW         | WLAN       | 0                              | 0                  |
| HIGH                | LOW         | BT         | 0                              | 0                  |
| LOW                 | LOW         | BT         | 1                              | 1                  |

To configure BT coexistence, enter the following commands:

1. Set WLAN traffic priority.  
`iwpriv wifi0 btcoex_wl_pri 0x34`

The following wlan priority bits and a bit set indicates particular WLAN traffic is of high priority.

- Best effort – Bit 0
- Background – Bit 1
- Video – Bit 2

- Voice – Bit 3
- Beacon – Bit 4
- Management– Bit 5
- $0x34 = 110100$  implies Video, Beacon and management traffic are high priority.
- Display current priority:  
`iwpriv wifi0 g_btcoex_wl_pri`
- Enable BT coex feature  
`iwpriv wifi0 btcoex_enable 1`
- Disable BT coex feature status:  
`iwpriv wifi0 btcoex_enable 0`
- Display current status:  
`iwpriv wifi0 g_btcoex_enable`
- Display current duty cycle parameter:  
`iwpriv wifi0 g_btcoex_dc`
- Set duty cycle parameters:  
`iwpriv wifi0 btcoex_dc <period> <wlan_duration>`  
Eg: `iwpriv wifi0 btcoex_dc 100 80`

## Configure CSRMesh

To configure CSRMesh, enter the following commands:

- Start Bluetopia PM:  
`export BTHOST_8311_SOC_TYPE=onboard  
SS1BTPM 1 /dev/ttyQHS0 115200 &`
- Start Mesh GW:  
`csrMeshGw -d0 -l3`
- Start Mesh test application:  
`csrMeshGwTestApp`
- Control light device sequence:  
`root@OpenWrt:~# csrMeshGwTestApp  
CsrMeshSockOpen: Creating UNIX Socket`  
`*****`

## Gateway Application :: Main Menu:

- 1 - Core
- 2 - Watchdog
- 3 - Config
- 4 - Group
- 5 - Sensor

6 - Actuator  
7 - Data  
8 - Bearer  
9 - Ping  
10 - Battery  
11 - Attention  
12 - Power  
13 - Light  
14 - LargeObjectTransfer  
15 - Action  
16 - Tuning  
17 - Time  
18 - Beacon  
19 - Beacon Proxy  
20 - Extension  
21 - Diagnostic  
22 - Asset  
23 - Tracker  
22 - Exit

\*\*\*\*\*

Enter Operation ID : 1

Core APIs

\*\*\*\*\*

### Gateway Application :: Core Menu

0 - Perform Authentication  
1 - Start Mesh  
2 - Stop Mesh  
3 - Reset Mesh  
4 - Get Network ID list

- 5 - Associate Gateway to Network
- 6 - Remove Network
- 7 - Get Local Device UUID
- 8 - Get Local Device ID
- 9 - Set Remote Device ID
- 10 - Set Network Key ID
- 11 - Set Network Key : Pass Phrase (pp1)
- 12 - Set Pass Phrase
- 13 - Find Mesh ID
- 14 - Find Network ID
- 15 - Register Sniffer
- 16 - UnRegister Sniffer
- 17 - Set Transmit State
- 18 - Get Transmit State
- 19 - Get Diagnostic Data Request
- 20 - Enable Diagnostic Data
- 21 - AddToBlacklist
- 22 - RemovefromBlackList
- 23 - Check the state of BlackListed Device
- 24 - LOT transfer
- 25 - LOT transfer cancel
- 26 - Set sniffer Modelmask
- 27 - Resume Authenticated Session
- 28 - Generate ID and CE challenge Keys
- 29 - Set TTL for ALL the networks
- 30 - Set Mesh Meta Data for MCP Packets
- 31 - Generate Session Key
- 32 - Register Extension Opcodes
- 33 - Send Extension Data Message

34 - Get device public key (This call is must prior to start association (22/23))  
 35 - Send Association Request with Auth Code (This will start association)  
 36 - Send Association Request with Confirmation Code (This will start association)  
 37 - Send confirmation Status (this api sends the confirmation buffer matching status to mesh for MASP resume)  
 38 - Init Mesh (Reset Mesh must be called prior to init)  
 39 - Send Key/IV Status Request  
 40 - Get Tracker Server Report  
 41 - Go to Main Menu  
 42 - Exit

---

```

/*****  

Enter Operation ID : 0  

CsrMeshAuthStart  

CsrMeshAuthStart, size: 69  

CsrSendMeshSecure  

Input:  

4300001e0080f42a9097a03fc58c620451b462cfb43d0ba5a45f393ffff1aa08f2818f3d5  

8c36b81d2d5939895c03bd85158ad3af2a0628bbd6c09c28b05c22ca28b57df35  

Output:  

4a0000000001e0080f42a9097a03fc58c620451b462cfb43d0ba5a45f393ffff1aa08f28  

18f3d58c36b81d2d5939895c03bd85158ad3af2a0628bbd6c09c28b05c22ca28b57df35  

00000000  

Rxd [70] bytes  

deserialize_msg:201  

Received Msg  

44000000000024800000eb60a4665e01e3b159f4c89b2417955cab5717f0cc45eb411cc8  

114347dffaa657f9451b6ca79c85c7b73f9031cb2f3ea7e7a8929fd0deab00000000  

msgClass=0 msgType=8024

>> CSR_MESH_AUTH_CFM_IND Received Status : SUCCESS

handleAuthChallenge
SB's type:2480
SB's status:0000
SB's key:  

eb60a4665e01e3b159f4c89b2417955cab5717f0cc45eb411cc8114347dffaa657f9451b  

6ca79c85c7b73f9031cb2f3e
SB's Public Ke:  

eb60a4665e01e3b159f4c89b2417955cab5717f0cc45eb411cc8114347dffaa657f9451b  

6ca79c85c7b73f9031cb2f3e
SB's Challenge:a7e7a8929fd0deab
Apps Secret:d33610f31ad4546dd94e5aa163ce5e6c10de31386a4dflef
Apps Challenge:89021a1c6211d47b
CsrMeshAuthConfirm
CsrSendMeshSecure
Input:0b00001f0089021a1c6211d47b
Output:1200010000001f0089021a1c6211d47b00000000

```

```

Rxd [ 30] bytes

deserialize_msg:201
Received Msg1c000100000025800000f54a24d4d7f6dd5bc70dcf48f319937400000000
msgClass=0 msgType=8025

>> CSR_MESH_AUTH_RANDOM_IND Received Status : SUCCESS

handleAuthRandomInd
App's Session:873eb9fa6125a9db7118e51dfc14dd8c
CsrMeshAuthEncryptStart
CsrSendMeshSecure
Input:
25000020004bd0e36731bd6528cf4892af1d9bef6261e31c20681cbc8423c2dc03714c
a4dd06
Output:
2c000200000020004bd0e36731bd6528cf4892af1d9bef6261e31c20681cbc8423c2dc
03714ca4dd0600000000
Rxd [ 32] bytes

deserialize_msg:201
Received Msg
1e00020000002680000036d1145b3acc365f24632d31e2dfb0789e1a00000000
msgClass=0 msgType=8026

>> CSR_MESH_AUTH_ENCR_START_IND Received Status : SUCCESS

handleAuthEncriInd
SB's Diversifier:36d1145b3acc365f
SB's IV:24632d31e2dfb0789e1a
SB's Session:238b00eb7a20aac2641765fcae5162d
CsrMeshAuthVerifyEncryption
CsrSendMeshSecure
Input:0300002100
Output:0a0003000077cd64b23ae32a
Rxd [ 14] bytes

deserialize_msg:201
Received Msg0c0003000031eb23d7aa598058c4
Decrypting Data
msgClass=0 msgType=8027

>> CSR_MESH_AUTH_VERIFY_ENCR_IND Received Status : SUCCESS

*****

```

## Gateway Application :: Core Menu

- 0 - Perform Authentication
- 1 - Start Mesh
- 2 - Stop Mesh
- 3 - Reset Mesh

- 4 - Get Network ID list
- 5 - Associate Gateway to Network
- 6 - Remove Network
- 7 - Get Local Device UUID
- 8 - Get Local Device ID
- 9 - Set Remote Device ID
- 10 - Set Network Key ID
- 11 - Set Network Key : Pass Phrase (pp1)
- 12 - Set Pass Phrase
- 13 - Find Mesh ID
- 14 - Find Network ID
- 15 - Register Sniffer
- 16 - UnRegister Sniffer
- 17 - Set Transmit State
- 18 - Get Transmit State
- 19 - Get Diagnostic Data Request
- 20 - Enable Diagnostic Data
- 21 - AddToBlacklist
- 22 - RemovefromBlackList
- 23 - Check the state of BlackListed Device
- 24 - LOT transfer
- 25 - LOT transfer cancel
- 26 - Set sniffer Modelmask
- 27 - Resume Authenticated Session
- 28 - Generate ID and CE challenge Keys
- 29 - Set TTL for ALL the networks
- 30 - Set Mesh Meta Data for MCP Packets
- 31 - Generate Session Key
- 32 - Register Extension Opcodes

- 33 - Send Extension Data Message
  - 34 - Get device public key (This call is must prior to start association (22/23))
  - 35 - Send Association Request with Auth Code (This will start association)
  - 36 - Send Association Request with Confirmation Code (This will start association)
  - 37 - Send confirmation Status (this api sends the confirmation buffer matching status to mesh for MASP resume)
  - 38 - Init Mesh (Reset Mesh must be called prior to init)
  - 39 - Send Key/IV Status Request
  - 40 - Get Tracker Server Report
  - 41 - Go to Main Menu
  - 42 - Exit
- \*\*\*\*\*
- ```

Enter Operation ID : 1

CsrSendMeshSecure
Input:0300000100
Output:0a000400009d6b7282d5fbee<<    Mesh Start

Rxd [14] bytes

deserialize_msg:201
Received Msg0c0004000082e9dbcb3f35c67be8
Decrypting Data
msgClass=0 msgType=8001

>>     CSRMESH_START_EVENT Received Status : SUCCESS

CsrSendMeshSecure
Input:03000a0000
Output:0a000500008b126a4b0b09e7
CsrSendMeshSecure
Input:03000a0100
Output:0a000600006056ec6a5d0665
CsrSendMeshSecure
Input:03000a0200
Output:0a000700001523d2fc01865f
CsrSendMeshSecure
Input:03000a0400
Output:0a0008000029283181258da2
CsrSendMeshSecure
Input:03000a0500
Output:0a000900005eb8c6f86255fb
CsrSendMeshSecure
Input:03000a0800
Output:0a000a0000496ee2e2d0749d
CsrSendMeshSecure

```

```

Input:03000a0b00
Output:0a000b0000f59717627b73ab
CsrSendMeshSecure
Input:03000a0c00
Output:0a000c0000991e0c6ca5989e
CsrSendMeshSecure
Input:03000a0d00
Output:0a000d0000238de3e92eefaa2
CsrSendMeshSecure
Input:03000a0e00
Output:0a000e00005f4bb8fe09b772
CsrSendMeshSecure
Input:03000a1300
Output:0a000f000016fa6f7e650464
CsrSendMeshSecure
Input:03000a1400
Output:0a00100000d56a067489d1e6
CsrSendMeshSecure
Input:03000a1e00
Output:0a00110000cff482aa1dc323
CsrSendMeshSecure
Input:03000a1c00
Output:0a00120000a7dbe58a64dfc7
CsrSendMeshSecure
Input:03000a1000
Output:0a00130000a5a60e85b0f8ae
CsrSendMeshSecure
Input:03000a2000
Output:0a001400001442f2613c470f
CsrSendMeshSecure
Input:03000a2100
Output:0a00150000089c31a62d9954
CsrSendMeshSecure
Input:03000a1d00
Output:0a00160000f2d2bace0dbb62
CsrSendMeshSecure
Input:03000a0a00
Output:0a00170000895babfb308481
CsrSendMeshSecure
Input:03000a2200
Output:0a0018000065172a901e1ef7
CsrSendMeshSecure
Input:03000a0600
Output:0a001900003579ba4537b2f8
CsrSendMeshSecure
Input:03000a0700
Output:0a001a0000cbe2be23420b6f

/*****************/

```

## Gateway Application :: Core Menu

- 0 - Perform Authentication
- 1 - Start Mesh

- 2 - Stop Mesh
- 3 - Reset Mesh
- 4 - Get Network ID list
- 5 - Associate Gateway to Network
- 6 - Remove Network
- 7 - Get Local Device UUID
- 8 - Get Local Device ID
- 9 - Set Remote Device ID
- 10 - Set Network Key ID
- 11 - Set Network Key : Pass Phrase (pp1)
- 12 - Set Pass Phrase
- 13 - Find Mesh ID
- 14 - Find Network ID
- 15 - Register Sniffer
- 16 - UnRegister Sniffer
- 17 - Set Transmit State
- 18 - Get Transmit State
- 19 - Get Diagnostic Data Request
- 20 - Enable Diagnostic Data
- 21 - AddToBlacklist
- 22 - RemovefromBlackList
- 23 - Check the state of BlackListed Device
- 24 - LOT transfer
- 25 - LOT transfer cancel
- 26 - Set sniffer Modelmask
- 27 - Resume Authenticated Session
- 28 - Generate ID and CE challenge Keys
- 29 - Set TTL for ALL the networks
- 30 - Set Mesh Meta Data for MCP Packets

31 - Generate Session Key  
 32 - Register Extension Opcodes  
 33 - Send Extension Data Message  
 34 - Get device public key (This call is must prior to start association (22/23))  
 35 - Send Association Request with Auth Code (This will start association)  
 36 - Send Association Request with Confirmation Code (This will start association)  
 37 - Send confirmation Status (this api sends the confirmation buffer matching status to mesh for MASP resume)  
 38 - Init Mesh (Reset Mesh must be called prior to init)  
 39 - Send Key/IV Status Request  
 40 - Get Tracker Server Report  
 41 - Go to Main Menu  
 42 - Exit

```

  *****/
  Enter Operation ID : 4

  CsrSendMeshSecure
  Input:0300000700
  Output:0a001b0000eba89f65818f04<<    Get Net Id List

  Rxd [20] bytes

  deserialize_msg:201
  Received Msg120005000082a9823a2d692e359a75aec9a6ed2d
  Decrypting Data
  msgClass=0 msgType=8007

  >>      CSR_MESH_NETWORK_ID_LIST_EVENT Received Status : SUCCESS
  NumOfNetIds = 0
  *****/

```

### Gateway Application :: Core Menu

- 0 - Perform Authentication
- 1 - Start Mesh
- 2 - Stop Mesh
- 3 - Reset Mesh
- 4 - Get Network ID list
- 5 - Associate Gateway to Network

- 6 - Remove Network
- 7 - Get Local Device UUID
- 8 - Get Local Device ID
- 9 - Set Remote Device ID
- 10 - Set Network Key ID
- 11 - Set Network Key : Pass Phrase (pp1)
- 12 - Set Pass Phrase
- 13 - Find Mesh ID
- 14 - Find Network ID
- 15 - Register Sniffer
- 16 - UnRegister Sniffer
- 17 - Set Transmit State
- 18 - Get Transmit State
- 19 - Get Diagnostic Data Request
- 20 - Enable Diagnostic Data
- 21 - AddToBlacklist
- 22 - RemovefromBlackList
- 23 - Check the state of BlackListed Device
- 24 - LOT transfer
- 25 - LOT transfer cancel
- 26 - Set sniffer Modelmask
- 27 - Resume Authenticated Session
- 28 - Generate ID and CE challenge Keys
- 29 - Set TTL for ALL the networks
- 30 - Set Mesh Meta Data for MCP Packets
- 31 - Generate Session Key
- 32 - Register Extension Opcodes
- 33 - Send Extension Data Message
- 34 - Get device public key (This call is must prior to start association (22/23))

- 35 - Send Association Request with Auth Code (This will start association)
- 36 - Send Association Request with Confirmation Code (This will start association)
- 37 - Send confirmation Status (this api sends the confirmation buffer matching status to mesh for MASP resume)
- 38 - Init Mesh (Reset Mesh must be called prior to init)
- 39 - Send Key/IV Status Request
- 40 - Get Tracker Server Report
- 41 - Go to Main Menu
- 42 - Exit

\*\*\*\*\*

```

Enter Operation ID : 12
Enter Local Device ID (in Hex form XXXX)   : 8888
Local Device ID : 0x8888
Enter the passPhrase : rajesh

CsrSendMeshSecure
Input:0c000005000672616a6573688888
Output:13001c00005dbbad8497086ae92e55eafa216b94b9<<    Set Pass phrase

Rxd [15] bytes

deserialize_msg:201
Received Msg0d000600005e4144e71ec0f60e9037
Decrypting Data
msgClass=0 msgType=8005

>>      CSR_MESH_SET_PASSPHRASE_EVENT Received Status : SUCCESS
NW_ID : 0x0

```

\*\*\*\*\*

## Gateway Application :: Core Menu

- 0 - Perform Authentication
- 1 - Start Mesh
- 2 - Stop Mesh
- 3 - Reset Mesh
- 4 - Get Network ID list
- 5 - Associate Gateway to Network
- 6 - Remove Network
- 7 - Get Local Device UUID

- 8 - Get Local Device ID
- 9 - Set Remote Device ID
- 10 - Set Network Key ID
- 11 - Set Network Key : Pass Phrase (pp1)
- 12 - Set Pass Phrase
- 13 - Find Mesh ID
- 14 - Find Network ID
- 15 - Register Sniffer
- 16 - UnRegister Sniffer
- 17 - Set Transmit State
- 18 - Get Transmit State
- 19 - Get Diagnostic Data Request
- 20 - Enable Diagnostic Data
- 21 - AddToBlacklist
- 22 - RemovefromBlackList
- 23 - Check the state of BlackListed Device
- 24 - LOT transfer
- 25 - LOT transfer cancel
- 26 - Set sniffer Modelmask
- 27 - Resume Authenticated Session
- 28 - Generate ID and CE challenge Keys
- 29 - Set TTL for ALL the networks
- 30 - Set Mesh Meta Data for MCP Packets
- 31 - Generate Session Key
- 32 - Register Extension Opcodes
- 33 - Send Extension Data Message
- 34 - Get device public key (This call is must prior to start association (22/23))
- 35 - Send Association Request with Auth Code (This will start association)
- 36 - Send Association Request with Confirmation Code (This will start association)

37 - Send confirmation Status (this api sends the confirmation buffer matching status to mesh for MASP resume)

38 - Init Mesh (Reset Mesh must be called prior to init)

39 - Send Key/IV Status Request

40 - Get Tracker Server Report

41 - Go to Main Menu

42 - Exit

\*\*\*\*\*

Enter Operation ID : 4

CsrSendMeshSecure

Input:0300000700

Output:0a001d0000bfec8617a2634<< Get Net Id List

Rxd [ 20 ] bytes

deserialize\_msg:201

Received Msg120007000091b6da9d52904d8de439f91bfc5f06

Decrypting Data

msgClass=0 msgType=8007

>> CSRMESH\_NETWORK\_ID\_LIST\_EVENT Received Status : SUCCESS

NumOfNetIds = 1

NetId[0] = 0

\*\*\*\*\*

## Gateway Application :: Core Menu

0 - Perform Authentication

1 - Start Mesh

2 - Stop Mesh

3 - Reset Mesh

4 - Get Network ID list

5 - Associate Gateway to Network

6 - Remove Network

7 - Get Local Device UUID

8 - Get Local Device ID

9 - Set Remote Device ID

10 - Set Network Key ID

- 11 - Set Network Key : Pass Phrase (pp1)
- 12 - Set Pass Phrase
- 13 - Find Mesh ID
- 14 - Find Network ID
- 15 - Register Sniffer
- 16 - UnRegister Sniffer
- 17 - Set Transmit State
- 18 - Get Transmit State
- 19 - Get Diagnostic Data Request
- 20 - Enable Diagnostic Data
- 21 - AddToBlacklist
- 22 - RemovefromBlackList
- 23 - Check the state of BlackListed Device
- 24 - LOT transfer
- 25 - LOT transfer cancel
- 26 - Set sniffer Modelmask
- 27 - Resume Authenticated Session
- 28 - Generate ID and CE challenge Keys
- 29 - Set TTL for ALL the networks
- 30 - Set Mesh Meta Data for MCP Packets
- 31 - Generate Session Key
- 32 - Register Extension Opcodes
- 33 - Send Extension Data Message
- 34 - Get device public key (This call is must prior to start association (22/23))
- 35 - Send Association Request with Auth Code (This will start association)
- 36 - Send Association Request with Confirmation Code (This will start association)
- 37 - Send confirmation Status (this api sends the confirmation buffer matching status to mesh for MASP resume)
- 38 - Init Mesh (Reset Mesh must be called prior to init)
- 39 - Send Key/IV Status Request

40 - Get Tracker Server Report

41 - Go to Main Menu

42 - Exit

\*\*\*\*\*

Enter Operation ID : 41

\*\*\*\*\*

### Gateway Application :: Main Menu :

1 - Core

2 - Watchdog

3 - Config

4 - Group

5 - Sensor

6 - Actuator

7 - Data

8 - Bearer

9 - Ping

10 - Battery

11 - Attention

12 - Power

13 - Light

14 - LargeObjectTransfer

15 - Action

16 - Tuning

17 - Time

18 - Beacon

19 - Beacon Proxy

20 - Extension

21 - Diagnostic

22 - Asset

23 - Tracker

22 - Exit

\*\*\*\*\*

Enter Operation ID : 13

Light Model

\*\*\*\*\*

### Gateway Application :: Light Menu :

1 - Set Remote Device ID

2 - Set Network Key ID

3 - Set Packet Repeat Value

4 - Set RGB with ACK

5 - Set RGB with NO ACK

6 - Set Light Level with ACK

7 - Set Light Level with NO ACK

8 - Set Power Level with Ack

9 - Set Power Level with NO Ack

10 - Set Color Temperature with Ack

11 - Get Light State

12 - Set Light White with ACK

13 - Set Light White with NO ACK

14 - Get Light White

15 - Go to Main Menu

16 - Exit

\*\*\*\*\*

Enter Operation ID : 1

Enter Remote Device ID (in Hex form XXXX) : 8001

Remote Device ID : 0x8001

\*\*\*\*\*

**Gateway Application :: Light Menu :**

- 1 - Set Remote Device ID
- 2 - Set Network Key ID
- 3 - Set Packet Repeat Value
- 4 - Set RGB with ACK
- 5 - Set RGB with NO ACK
- 6 - Set Light Level with ACK
- 7 - Set Light Level with NO ACK
- 8 - Set Power Level with Ack
- 9 - Set Power Level with NO Ack
- 10 - Set Color Temperature with Ack
- 11 - Get Light State
- 12 - Set Light White with ACK
- 13 - Set Light White with NO ACK
- 14 - Get Light White
- 15 - Go to Main Menu
- 16 - Exit

\*\*\*\*\*

Enter Operation ID : 4

Enter Color :: 1: RED, 2: GREEN, 3: BLUE : 2

```
CsrSendMeshSecure
Input:0f000b038a000180010a00ff0005000001
Output:16001e0000ffe8c3b9a0c6283a645042b150c61b871112d8
Rxd [31] bytes

deserialize_msg:201
Received Msg
1d0008000014baa7f5cb2ab7ebbd37ff97443e53da274e34dba148f18f1310
Decrypting Data
msgClass=b msgType=8a08
>> Message SeqNo=0xea7c2,
>> LIGHT_STATE Received
>> NW_ID : 0x0, SRC_ID : 0x8001, DST_ID : 0x8888, P : 1, L : a, R :
ff, G : 0, B : 0, CD : 0, S : 0, Tid : 0
```

\*\*\*\*\*

## Gateway Application :: Light Menu :

- 1 - Set Remote Device ID
- 2 - Set Network Key ID
- 3 - Set Packet Repeat Value
- 4 - Set RGB with ACK
- 5 - Set RGB with NO ACK
- 6 - Set Light Level with ACK
- 7 - Set Light Level with NO ACK
- 8 - Set Power Level with Ack
- 9 - Set Power Level with NO Ack
- 10 - Set Color Temperature with Ack
- 11 - Get Light State
- 12 - Set Light White with ACK
- 13 - Set Light White with NO ACK
- 14 - Get Light White
- 15 - Go to Main Menu
- 16 - Exit

\*\*\*\*\*

Enter Operation ID : 4

Enter Color :: 1: RED, 2: GREEN, 3: BLUE : 3

```
CsrSendMeshSecure
Input:0f000b038a000180010a0000ff05000101
Output:16001f00001f6fe9458d5d09702fc03d0f72bc07837497a1
Rxd [31] bytes

deserialize_msg:201
Received Msg
1d00090000015a5e93cbe15776f2ec55729fc9398315aa9975bb671c0c995b
Decrypting Data
msgClass=b msgType=8a09

>> Message SeqNo=0xea7c3,
>> LIGHT_STATE_NO_ACK Received
>> NW_ID : 0x0, SRC_ID : 0x8001, DST_ID : 0x8888, P : 1, L : a, R :
0, G : ff, B : 0, CD : 0, S : 0, Tid : 0

*****
```

**Establishing GATT connection with 1010 device:**

```

./LinuxGATM_CLT

Initialize 1
SetDevicePower 1 - /*use this command only if mesh GW is not started.*/
DisconnectRemoteDevice 00025B0045D7 1 - /* To disconnect the device */
ConnectWithRemoteDevice 00025B0045D7 1
RegisterGATTCallback
WriteValue 00025B0045D7 0x001F 0 1 - /* write to GATT server in 1010
device.*/

```

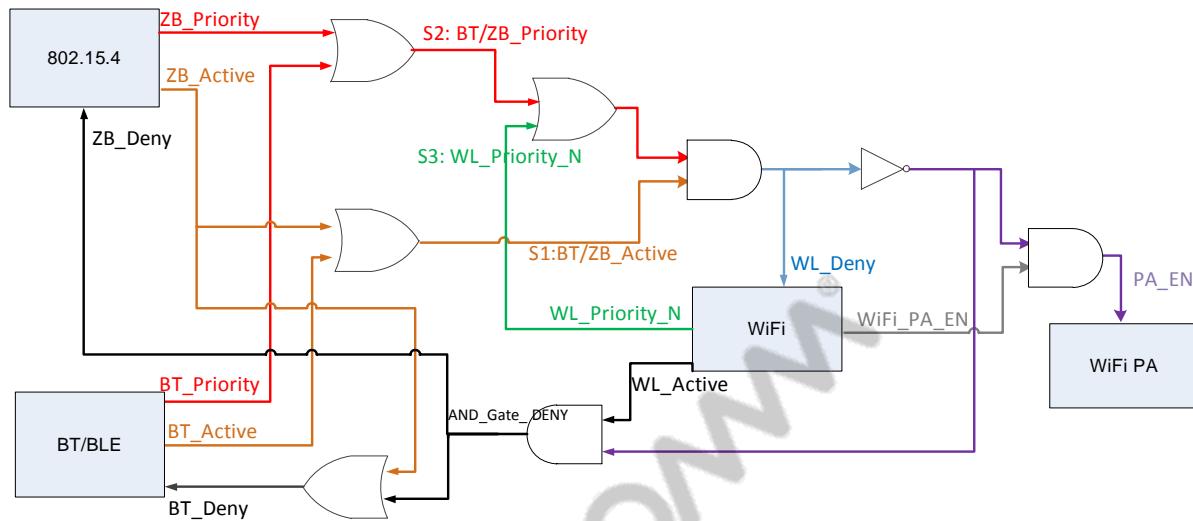
## 20.4 Coexistence of WLAN and ZigBee for CSRMesh

Coexistence of WLAN and Zigbee is implemented. This interconnection of ZigBee and WLAN enables transmission without any interference between the two components. Zigbee high priority traffic cannot be overridden by WLAN.

### Hardware interfaces for IPQ4019

- ZB active and ZB priority GPIO from ZB module and WLAN priority from WLAN module used to intimate that respective module is transmitting and used for co-existence to stop ZB/WLAN so that other traffic and flow through without interference issues.
- Zigbee module asserts ZB active GPIO when it has to send ZB traffic, it also asserts ZB priority GPIO if the ZB traffic is high priority.
- WLAN module de-asserts WLAN priority GPIO if there is high priority WLAN traffic and assert WLAN priority GPIO if WLAN traffic is low priority.
- ZB priority and WLAN priority GPIOs are ORed and output is ANDed with ZB active to generate LTE-ACTIVE which is fed as input to WLAN.
- If LTE-ACTIVE GPIO goes high, WLAN tx will be stopped and
- ZB traffic will flow through.
- WLAN ACTIVE is ANDed with NOT of LTE-ACTIVE and fed as
- ZB-DENY to ZB module to stop ZB traffic.

The following hardware logic ensures WLAN traffic is stopped if there is high priority ZB priority and stops ZB low priority traffic only when there is high priority WLAN traffic:



Allow ZB high priority traffic irrespective of WLAN priority.

\* Allow WLAN high priority traffic only when ZB traffic priority is low.

\* Allow ZB low/high priority traffic when WLAN priority is low.

P1:ZB/ZB_ACTIVE	P2:ZB/ZB_Priority	P3:WL_Priority_N	Priority
0	0	0	WLAN TX goes through
1	0	1	Stop WLAN TX
1	1	1	Stop WLAN TX
1	0	0	WLAN TX goes through
1	1	0	Stop WLAN TX

### Configure ZigBee coexistence

To configure ZigBee coexistence, enter the following commands:

1. Set WLAN traffic priority.

```
iwpriv wifi0 btcoex_wl_pri 0x34
```

The following wlan priority bits and a bit set indicates particular WLAN traffic is of high priority.

Best effort – Bit 0, Background – Bit 1, Video – Bit 2, Voice – Bit 3, Beacon – Bit 4, Management– Bit 5

0x34 = 110100 implies Video, Beacon and management traffic are high priority.

2. Display current priority:

```
iwpriv wifi0 g_btcoex_wl_pri
```

3. Enable ZigBee coex feature

```
iwpriv wifi0 btcoex_enable 1
```

## 4. Disable ZigBee coex feature:

```
iwpriv wifi0 btcoex_enable 0
```

## 5. Display current status:

```
iwpriv wifi0 g_btcoex_enable
```

## 6. Enter the following ZigBee commands:

```
echo "44" > /sys/class/gpio/export
echo "46" > /sys/class/gpio/export
echo "47" > /sys/class/gpio/export
echo "31" > /sys/class/gpio/export
echo "42" > /sys/class/gpio/export
echo "49" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio49/direction
echo "1" > /sys/class/gpio/gpio49/value
echo "45" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio45/direction
echo "1" > /sys/class/gpio/gpio45/value
XncpLedHost-SPI6
network find unused
network pjoin 0xff
```

## 7. turn on the bulb and wait for few seconds

```
option disc 0x104 0x0006
zcl on-off toggle
send 0x2B14 1 1
```

**Configure CSRMesh**

To configure CSRMesh, enter the following commands:

## 1. Start Bluetopia PM:

```
export BTHOST_8311_SOC_TYPE=onboard
SS1BTPM 1 /dev/ttyQHS0 115200 &
```

## 2. Start Mesh GW:

```
csrMeshGw -d0 -13
```

## 3. Start Mesh test application:

```
csrMeshGwTestApp
```

## 4. Control light device sequence:

```
root@OpenWrt:~# csrMeshGwTestApp
CsrMeshSockOpen: Creating UNIX Socket
```

```
*****
```

**Gateway Application :: Main Menu :**

1 - Core

2 - Watchdog

3 - Config

4 - Group

5 - Sensor  
6 - Actuator  
7 - Data  
8 - Bearer  
9 - Ping  
10 - Battery  
11 - Attention  
12 - Power  
13 - Light  
14 - LargeObjectTransfer  
15 - Action  
16 - Tuning  
17 - Time  
18 - Beacon  
19 - Beacon Proxy  
20 - Extension  
21 - Diagnostic  
22 - Asset  
23 - Tracker  
22 - Exit

\*\*\*\*\*

Enter Operation ID : 1

Core APIs

\*\*\*\*\*

### Gateway Application :: Core Menu

0 - Perform Authentication  
1 - Start Mesh  
2 - Stop Mesh  
3 - Reset Mesh

- 4 - Get Network ID list
- 5 - Associate Gateway to Network
- 6 - Remove Network
- 7 - Get Local Device UUID
- 8 - Get Local Device ID
- 9 - Set Remote Device ID
- 10 - Set Network Key ID
- 11 - Set Network Key : Pass Phrase (pp1)
- 12 - Set Pass Phrase
- 13 - Find Mesh ID
- 14 - Find Network ID
- 15 - Register Sniffer
- 16 - UnRegister Sniffer
- 17 - Set Transmit State
- 18 - Get Transmit State
- 19 - Get Diagnostic Data Request
- 20 - Enable Diagnostic Data
- 21 - AddToBlacklist
- 22 - RemovefromBlackList
- 23 - Check the state of BlackListed Device
- 24 - LOT transfer
- 25 - LOT transfer cancel
- 26 - Set sniffer Modelmask
- 27 - Resume Authenticated Session
- 28 - Generate ID and CE challenge Keys
- 29 - Set TTL for ALL the networks
- 30 - Set Mesh Meta Data for MCP Packets
- 31 - Generate Session Key
- 32 - Register Extension Opcodes

33 - Send Extension Data Message  
 34 - Get device public key (This call is must prior to start association (22/23))  
 35 - Send Association Request with Auth Code (This will start association)  
 36 - Send Association Request with Confirmation Code (This will start association)  
 37 - Send confirmation Status (this api sends the confirmation buffer matching status to mesh for MASP resume)  
 38 - Init Mesh (Reset Mesh must be called prior to init)  
 39 - Send Key/IV Status Request  
 40 - Get Tracker Server Report  
 41 - Go to Main Menu  
 42 - Exit

```

/*****
Enter Operation ID : 0
CsrMeshAuthStart
CsrMeshAuthStart, size: 69
CsrSendMeshSecure
Input:
4300001e0080f42a9097a03fc58c620451b462cfb43d0ba5a45f393fff1aa08f2818f3d5
8c36b81d2d5939895c03bd85158ad3af2a0628bbd6c09c28b05c22ca28b57df35
Output:
4a0000000001e0080f42a9097a03fc58c620451b462cfb43d0ba5a45f393fff1aa08f28
18f3d58c36b81d2d5939895c03bd85158ad3af2a0628bbd6c09c28b05c22ca28b57df35
00000000
Rxd [70] bytes
deserialize_msg:201
Received Msg
44000000000024800000eb60a4665e01e3b159f4c89b2417955cab5717f0cc45eb411cc8
114347dffaa657f9451b6ca79c85c7b73f9031cb2f3ea7e7a8929fd0deab00000000
msgClass=0 msgType=8024

>> CSR_MESH_AUTH_CFM_IND Received Status : SUCCESS

handleAuthChallenge
SB's type:2480
SB's status:0000
SB's key:
eb60a4665e01e3b159f4c89b2417955cab5717f0cc45eb411cc8114347dffaa657f9451b
6ca79c85c7b73f9031cb2f3e
SB's Public Ke:
eb60a4665e01e3b159f4c89b2417955cab5717f0cc45eb411cc8114347dffaa657f9451b
6ca79c85c7b73f9031cb2f3e
SB's Challenge:a7e7a8929fd0deab
Apps Secret:d33610f31ad4546dd94e5aa163ce5e6c10de31386a4df1ef
Apps Challenge:89021a1c6211d47b
CsrMeshAuthConfirm
CsrSendMeshSecure

```

```

Input:0b00001f0089021a1c6211d47b
Output:1200010000001f0089021a1c6211d47b00000000
Rxd [30] bytes

deserialize_msg:201
Received Msg1c000100000025800000f54a24d4d7f6dd5bc70dcf48f319937400000000
msgClass=0 msgType=8025

>> CSR_MESH_AUTH_RANDOM_IND Received Status : SUCCESS

handleAuthRandomInd
App's Session:873eb9fa6125a9db7118e51dfc14dd8c
CsrMeshAuthEncryptStart
CsrSendMeshSecure
Input:
25000020004bd0e36731bd6528cf4892af1d9bef6261e31c20681cbc8423c2dc03714c
a4dd06
Output:
2c000200000020004bd0e36731bd6528cf4892af1d9bef6261e31c20681cbc8423c2dc
03714ca4dd0600000000
Rxd [32] bytes
deserialize_msg:201
Received Msg
1e00020000002680000036d1145b3acc365f24632d31e2dfb0789e1a00000000
msgClass=0 msgType=8026
>> CSR_MESH_AUTH_ENCR_START_IND Received Status : SUCCESS
handleAuthEncriInd
SB's Diversifier:36d1145b3acc365f
SB's IV:24632d31e2dfb0789e1a
SB's Session:238b00eb7a20aaac2641765fcae5162d
CsrMeshAuthVerifyEncryption
CsrSendMeshSecure
Input:0300002100
Output:0a0003000077cd64b23ae32a
Rxd [14] bytes
deserialize_msg:201
Received Msg0c0003000031eb23d7aa598058c4
Decrypting Data
msgClass=0 msgType=8027
>> CSR_MESH_AUTH_VERIFY_ENCR_IND Received Status : SUCCESS

/*****

```

## Gateway Application :: Core Menu

- 0 - Perform Authentication
- 1 - Start Mesh
- 2 - Stop Mesh
- 3 - Reset Mesh
- 4 - Get Network ID list
- 5 - Associate Gateway to Network

- 6 - Remove Network
- 7 - Get Local Device UUID
- 8 - Get Local Device ID
- 9 - Set Remote Device ID
- 10 - Set Network Key ID
- 11 - Set Network Key : Pass Phrase (pp1)
- 12 - Set Pass Phrase
- 13 - Find Mesh ID
- 14 - Find Network ID
- 15 - Register Sniffer
- 16 - UnRegister Sniffer
- 17 - Set Transmit State
- 18 - Get Transmit State
- 19 - Get Diagnostic Data Request
- 20 - Enable Diagnostic Data
- 21 - AddToBlacklist
- 22 - RemovefromBlackList
- 23 - Check the state of BlackListed Device
- 24 - LOT transfer
- 25 - LOT transfer cancel
- 26 - Set sniffer Modelmask
- 27 - Resume Authenticated Session
- 28 - Generate ID and CE challenge Keys
- 29 - Set TTL for ALL the networks
- 30 - Set Mesh Meta Data for MCP Packets
- 31 - Generate Session Key
- 32 - Register Extension Opcodes
- 33 - Send Extension Data Message
- 34 - Get device public key(This call is must prior to start association (22/23))

- 35 - Send Association Request with Auth Code (This will start association)
- 36 - Send Association Request with Confirmation Code (This will start association)
- 37 - Send confirmation Status (this api sends the confirmation buffer matching status to mesh for MASP resume)
- 38 - Init Mesh (Reset Mesh must be called prior to init)
- 39 - Send Key/IV Status Request
- 40 - Get Tracker Server Report
- 41 - Go to Main Menu
- 42 - Exit

\*\*\*\*\*

```

Enter Operation ID : 1
CsrSendMeshSecure
Input:0300000100
Output:0a000400009d6b7282d5fbee<<      Mesh Start
Rxd [14] bytes
deserialize_msg:201
Received Msg0c0004000082e9dbcb3f35c67be8
Decrypting Data
msgClass=0 msgType=8001
>>    CSRMESH_START_EVENT Received Status : SUCCESS
CsrSendMeshSecure
Input:03000a0000
Output:0a000500008b126a4b0b09e7
CsrSendMeshSecure
Input:03000a0100
Output:0a000600006056ec6a5d0665
CsrSendMeshSecure
Input:03000a0200
Output:0a000700001523d2fc01865f
CsrSendMeshSecure
Input:03000a0400
Output:0a0008000029283181258da2
CsrSendMeshSecure
Input:03000a0500
Output:0a000900005eb8c6f86255fb
CsrSendMeshSecure
Input:03000a0800
Output:0a000a0000496ee2e2d0749d
CsrSendMeshSecure
Input:03000a0b00
Output:0a000b0000f59717627b73ab
CsrSendMeshSecure
Input:03000a0c00
Output:0a000c0000991e0c6ca5989e
CsrSendMeshSecure
Input:03000a0d00
Output:0a000d0000238de3e92eefaa2
CsrSendMeshSecure

```

```

Input:03000a0e00
Output:0a000e00005f4bb8fe09b772
CsrSendMeshSecure
Input:03000a1300
Output:0a000f000016fa6f7e650464
CsrSendMeshSecure
Input:03000a1400
Output:0a00100000d56a067489d1e6
CsrSendMeshSecure
Input:03000a1e00
Output:0a00110000cff482aa1dc323
CsrSendMeshSecure
Input:03000a1c00
Output:0a00120000a7dbe58a64dfc7
CsrSendMeshSecure
Input:03000a1000
Output:0a00130000a5a60e85b0f8ae
CsrSendMeshSecure
Input:03000a2000
Output:0a001400001442f2613c470f
CsrSendMeshSecure
Input:03000a2100
Output:0a00150000089c31a62d9954
CsrSendMeshSecure
Input:03000a1d00
Output:0a00160000f2d2bace0dbb62
CsrSendMeshSecure
Input:03000a0a00
Output:0a00170000895babfb308481
CsrSendMeshSecure
Input:03000a2200
Output:0a0018000065172a901e1ef7
CsrSendMeshSecure
Input:03000a0600
Output:0a001900003579ba4537b2f8
CsrSendMeshSecure
Input:03000a0700
Output:0a001a0000cbe2be23420b6f

/*****************/

```

## Gateway Application :: Core Menu

- 0 - Perform Authentication
- 1 - Start Mesh
- 2 - Stop Mesh
- 3 - Reset Mesh
- 4 - Get Network ID list
- 5 - Associate Gateway to Network
- 6 - Remove Network

- 7 - Get Local Device UUID
- 8 - Get Local Device ID
- 9 - Set Remote Device ID
- 10 - Set Network Key ID
- 11 - Set Network Key : Pass Phrase (pp1)
- 12 - Set Pass Phrase
- 13 - Find Mesh ID
- 14 - Find Network ID
- 15 - Register Sniffer
- 16 - UnRegister Sniffer
- 17 - Set Transmit State
- 18 - Get Transmit State
- 19 - Get Diagnostic Data Request
- 20 - Enable Diagnostic Data
- 21 - AddToBlacklist
- 22 - RemovefromBlackList
- 23 - Check the state of BlackListed Device
- 24 - LOT transfer
- 25 - LOT transfer cancel
- 26 - Set sniffer Modelmask
- 27 - Resume Authenticated Session
- 28 - Generate ID and CE challenge Keys
- 29 - Set TTL for ALL the networks
- 30 - Set Mesh Meta Data for MCP Packets
- 31 - Generate Session Key
- 32 - Register Extension Opcodes
- 33 - Send Extension Data Message
- 34 - Get device public key(This call is must prior to start association (22/23))
- 35 - Send Association Request with Auth Code (This will start association)

- 36 - Send Association Request with Confirmation Code (This will start association)
- 37 - Send confirmation Status (this api sends the confirmation buffer matching status to mesh for MASP resume)
- 38 - Init Mesh (Reset Mesh must be called prior to init)
- 39 - Send Key/IV Status Request
- 40 - Get Tracker Server Report
- 41 - Go to Main Menu
- 42 - Exit

```
/*****************/
Enter Operation ID : 4
CsrSendMeshSecure
Input:0300000700
Output:0a001b0000eba89f65818f04<<    Get Net Id List
Rxd [20] bytes
deserialize_msg:201
Received Msg120005000082a9823a2d692e359a75aec9a6ed2d
Decrypting Data
msgClass=0 msgType=8007
>>      CSRMEASH_NETWORK_ID_LIST_EVENT Received Status : SUCCESS
        NumOfNetIds = 0
/*****************/
```

## Gateway Application :: Core Menu

- 0 - Perform Authentication
- 1 - Start Mesh
- 2 - Stop Mesh
- 3 - Reset Mesh
- 4 - Get Network ID list
- 5 - Associate Gateway to Network
- 6 - Remove Network
- 7 - Get Local Device UUID
- 8 - Get Local Device ID
- 9 - Set Remote Device ID
- 10 - Set Network Key ID
- 11 - Set Network Key : Pass Phrase (pp1)
- 12 - Set Pass Phrase

- 13 - Find Mesh ID
- 14 - Find Network ID
- 15 - Register Sniffer
- 16 - UnRegister Sniffer
- 17 - Set Transmit State
- 18 - Get Transmit State
- 19 - Get Diagnostic Data Request
- 20 - Enable Diagnostic Data
- 21 - AddToBlacklist
- 22 - RemovefromBlackList
- 23 - Check the state of BlackListed Device
- 24 - LOT transfer
- 25 - LOT transfer cancel
- 26 - Set sniffer Modelmask
- 27 - Resume Authenticated Session
- 28 - Generate ID and CE challenge Keys
- 29 - Set TTL for ALL the networks
- 30 - Set Mesh Meta Data for MCP Packets
- 31 - Generate Session Key
- 32 - Register Extension Opcodes
- 33 - Send Extension Data Message
- 34 - Get device public key (This call is must prior to start association (22/23))
- 35 - Send Association Request with Auth Code (This will start association)
- 36 - Send Association Request with Confirmation Code (This will start association)
- 37 - Send confirmation Status (this api sends the confirmation buffer matching status to mesh for MASP resume)
- 38 - Init Mesh (Reset Mesh must be called prior to init)
- 39 - Send Key/IV Status Request
- 40 - Get Tracker Server Report
- 41 - Go to Main Menu

42 - Exit

```
*****
Enter Operation ID : 12
Enter Local Device ID (in Hex form XXXX)  : 8888
Local Device ID : 0x8888
Enter the passPhrase : rajesh
CsrSendMeshSecure
Input:0c000005000672616a6573688888
Output:13001c00005dbbad8497086ae92e55eafa216b94b9<<      Set Pass phrase
Rxd [15] bytes
deserialize_msg:201
Received Msg0d000600005e4144e71ec0f60e9037
Decrypting Data
msgClass=0 msgType=8005

>>      CSRMEASH_SET_PASSPHRASE_EVENT Received Status : SUCCESS
NW_ID : 0x0
*****
```

## Gateway Application :: Core Menu

- 0 - Perform Authentication
- 1 - Start Mesh
- 2 - Stop Mesh
- 3 - Reset Mesh
- 4 - Get Network ID list
- 5 - Associate Gateway to Network
- 6 - Remove Network
- 7 - Get Local Device UUID
- 8 - Get Local Device ID
- 9 - Set Remote Device ID
- 10 - Set Network Key ID
- 11 - Set Network Key : Pass Phrase (pp1)
- 12 - Set Pass Phrase
- 13 - Find Mesh ID
- 14 - Find Network ID
- 15 - Register Sniffer
- 16 - UnRegister Sniffer

- 17 - Set Transmit State
- 18 - Get Transmit State
- 19 - Get Diagnostic Data Request
- 20 - Enable Diagnostic Data
- 21 - AddToBlacklist
- 22 - RemovefromBlackList
- 23 - Check the state of BlackListed Device
- 24 - LOT transfer
- 25 - LOT transfer cancel
- 26 - Set sniffer Modelmask
- 27 - Resume Authenticated Session
- 28 - Generate ID and CE challenge Keys
- 29 - Set TTL for ALL the networks
- 30 - Set Mesh Meta Data for MCP Packets
- 31 - Generate Session Key
- 32 - Register Extension Opcodes
- 33 - Send Extension Data Message
- 34 - Get device public key(This call is must prior to start association (22/23))
- 35 - Send Association Request with Auth Code (This will start association)
- 36 - Send Association Request with Confirmation Code (This will start association)
- 37 - Send confirmation Status (this api sends the confirmation buffer matching status to mesh for MASP resume)
- 38 - Init Mesh (Reset Mesh must be called prior to init)
- 39 - Send Key/IV Status Request
- 40 - Get Tracker Server Report
- 41 - Go to Main Menu
- 42 - Exit

```
*****
Enter Operation ID : 4
CsrSendMeshSecure
Input:0300000700
Output:0a001d0000bfec8617a2634<<    Get Net Id List
```

```
Rxd [20] bytes
deserialize_msg:201
Received Msg120007000091b6da9d52904d8de439f91bfc5f06
Decrypting Data
msgClass=0 msgType=8007
>>     CSRMEASH_NETWORK_ID_LIST_EVENT Received Status : SUCCESS
NumOfNetIds = 1
NetId[0] = 0

/*****
```

## Gateway Application :: Core Menu

- 0 - Perform Authentication
- 1 - Start Mesh
- 2 - Stop Mesh
- 3 - Reset Mesh
- 4 - Get Network ID list
- 5 - Associate Gateway to Network
- 6 - Remove Network
- 7 - Get Local Device UUID
- 8 - Get Local Device ID
- 9 - Set Remote Device ID
- 10 - Set Network Key ID
- 11 - Set Network Key : Pass Phrase (pp1)
- 12 - Set Pass Phrase
- 13 - Find Mesh ID
- 14 - Find Network ID
- 15 - Register Sniffer
- 16 - UnRegister Sniffer
- 17 - Set Transmit State
- 18 - Get Transmit State
- 19 - Get Diagnostic Data Request
- 20 - Enable Diagnostic Data
- 21 - AddToBlacklist
- 22 - RemovefromBlackList

- 23 - Check the state of BlackListed Device
- 24 - LOT transfer
- 25 - LOT transfer cancel
- 26 - Set sniffer Modelmask
- 27 - Resume Authenticated Session
- 28 - Generate ID and CE challenge Keys
- 29 - Set TTL for ALL the networks
- 30 - Set Mesh Meta Data for MCP Packets
- 31 - Generate Session Key
- 32 - Register Extension Opcodes
- 33 - Send Extension Data Message
- 34 - Get device public key (This call is must prior to start association (22/23))
- 35 - Send Association Request with Auth Code (This will start association)
- 36 - Send Association Request with Confirmation Code (This will start association)
- 37 - Send confirmation Status (this api sends the confirmation buffer matching status to mesh for MASP resume)
- 38 - Init Mesh (Reset Mesh must be called prior to init)
- 39 - Send Key/IV Status Request
- 40 - Get Tracker Server Report
- 41 - Go to Main Menu
- 42 - Exit

\*\*\*\*\*

Enter Operation ID : 41

\*\*\*\*\*

### Gateway Application :: Main Menu :

- 1 - Core
- 2 - Watchdog
- 3 - Config
- 4 - Group
- 5 - Sensor

6 - Actuator  
7 - Data  
8 - Bearer  
9 - Ping  
10 - Battery  
11 - Attention  
12 - Power  
13 - Light  
14 - LargeObjectTransfer  
15 - Action  
16 - Tuning  
17 - Time  
18 - Beacon  
19 - Beacon Proxy  
20 - Extension  
21 - Diagnostic  
22 - Asset  
23 - Tracker  
22 - Exit

\*\*\*\*\*

Enter Operation ID : 13

Light Model

\*\*\*\*\*

### Gateway Application :: Light Menu :

- 1 - Set Remote Device ID
- 2 - Set Network Key ID
- 3 - Set Packet Repeat Value
- 4 - Set RGB with ACK
- 5 - Set RGB with NO ACK

- 6 - Set Light Level with ACK
- 7 - Set Light Level with NO ACK
- 8 - Set Power Level with Ack
- 9 - Set Power Level with NO Ack
- 10 - Set Color Temperature with Ack
- 11 - Get Light State
- 12 - Set Light White with ACK
- 13 - Set Light White with NO ACK
- 14 - Get Light White
- 15 - Go to Main Menu
- 16 - Exit

```
*****  
Enter Operation ID : 1  
Enter Remote Device ID (in Hex form XXXX) : 8001  
Remote Device ID : 0x8001
```

```
*****
```

### Gateway Application :: Light Menu :

- 1 - Set Remote Device ID
- 2 - Set Network Key ID
- 3 - Set Packet Repeat Value
- 4 - Set RGB with ACK
- 5 - Set RGB with NO ACK
- 6 - Set Light Level with ACK
- 7 - Set Light Level with NO ACK
- 8 - Set Power Level with Ack
- 9 - Set Power Level with NO Ack
- 10 - Set Color Temperature with Ack
- 11 - Get Light State
- 12 - Set Light White with ACK
- 13 - Set Light White with NO ACK

14 - Get Light White

15 - Go to Main Menu

16 - Exit

```
*****
Enter Operation ID : 4
Enter Color :: 1: RED, 2: GREEN, 3: BLUE   : 2
CsrSendMeshSecure
Input:0f000b038a000180010a00ff0005000001
Output:16001e0000ffe8c3b9a0c6283a645042b150c61b871112d8
Rxd [31] bytes
deserialize_msg:201
Received Msg
1d0008000014baa7f5cb2ab7ebbd37ff97443e53da274e34dba148f18f1310
Decrypting Data
msgClass=b msgType=8a08

>>     Message SeqNo=0xea7c2,
>>     LIGHT_STATE Received
>>     NW_ID : 0x0, SRC_ID : 0x8001, DST_ID : 0x8888, P : 1, L : a, R :
ff, G : 0, B : 0, CD : 0, S : 0, Tid : 0
*****
```

### **Gateway Application :: Light Menu :**

- 1 - Set Remote Device ID
- 2 - Set Network Key ID
- 3 - Set Packet Repeat Value
- 4 - Set RGB with ACK
- 5 - Set RGB with NO ACK
- 6 - Set Light Level with ACK
- 7 - Set Light Level with NO ACK
- 8 - Set Power Level with Ack
- 9 - Set Power Level with NO Ack
- 10 - Set Color Temperature with Ack
- 11 - Get Light State
- 12 - Set Light White with ACK
- 13 - Set Light White with NO ACK
- 14 - Get Light White
- 15 - Go to Main Menu

16 - Exit

```
*****
Enter Operation ID : 4
Enter Color :: 1: RED, 2: GREEN, 3: BLUE   : 3
CsrSendMeshSecure
Input:0f000b038a000180010a0000ff05000101
Output:16001f00001f6fe9458d5d09702fc03d0f72bc07837497a1
Rxd [31] bytes
deserialize_msg:201
Received Msg
1d00090000015a5e93cbe15776f2ec55729fc9398315aa9975bb671c0c995b
Decrypting Data
msgClass=b msgType=8a09
>> Message SeqNo=0xea7c3,
>> LIGHT_STATE_NO_ACK Received
>> NW_ID : 0x0, SRC_ID : 0x8001, DST_ID : 0x8888, P : 1, L : a, R :
0, G : ff, B : 0, CD : 0, S : 0, Tid : 0
*****
```

## 20.5 Coexistence of Bluetooth and ZigBee with WLAN on IPQ8064

This section describes the coexistence of Bluetooth (BT) and ZigBee with WLAN on IPQ8064 and QCA9984. The connection between the WLAN and BT is using a 2.5-wire interface. This interconnection of BT and ZigBee with WLAN enables transmission without any interference between the two components. The time division multiplexing (TDM) mechanism is enhanced to support coexistence of BT and ZigBee with WLAN. After the BT coexistence commands and BT commands and BT commands are configured on the two devices, namely IPQ8064 and QCA9984, as the IoT Central platforms, load the firmware image and restart WLAN.

### 20.5.1 Behavior of hardware interfaces for Bluetooth/Zigbee and WLAN coexistence

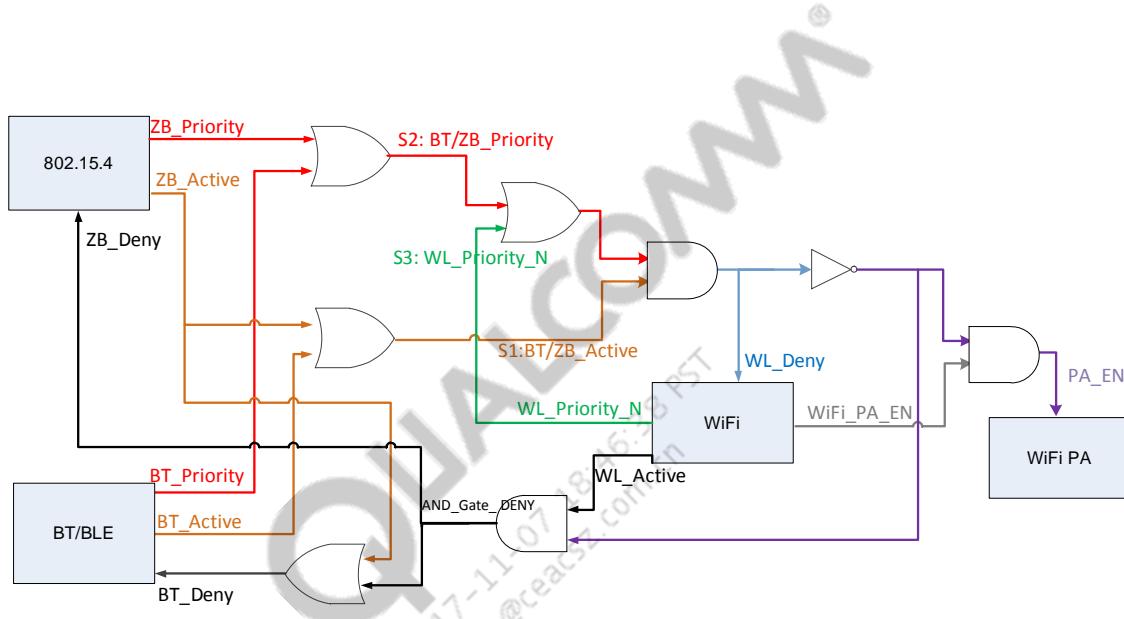
BT active and BT priority GPIO from BT module and WLAN priority from WLAN module are used to indicate that the respective module is transmitting. These GPIO signals signify that the corresponding module is used for coexistence to stop BT/WLAN to enable other traffic to flow without interference problems. BT module asserts BT active GPIO when it must send BT traffic. This module also asserts BT priority GPIO if the BT traffic is of high priority. WLAN module de-asserts WLAN priority GPIO if high-priority WLAN traffic is present, and asserts WLAN priority GPIO if WLAN traffic is of low priority.

BT priority and WLAN priority GPIOs are ORed and output is ANDed with BT active to generate LTE-ACTIVE, which is forwarded as an input to WLAN. If LTE-ACTIVE GPIO becomes high, WLAN Tx is stopped and BT traffic continues to flow through. WLAN ACTIVE is ANDed with NOT of LTE-ACTIVE and forwarded as BT-DENY to BT module to stop BT traffic.

The following hardware logic ensures that WLAN traffic is stopped if high-priority BT traffic is present, and that low-priority BT traffic is stopped only when high-priority WLAN traffic is present:

- Allow BT high priority traffic irrespective of WLAN priority.
- Allow WLAN high priority traffic only when BT traffic priority is low.
- Allow BT low/high priority traffic when WLAN priority is low.

The following figure shows the interconnection of the hardware interfaces and the priority values:



Traffic transmission is permitted based on the following guidelines:

P1:BT/ZB_ACTIVE	P2:BT/ZB_Priority	P3:WL_Priority_N	Priority
0	0	0	WLAN TX goes through
1	0	1	Stop WLAN TX
1	1	1	Stop WLAN TX
1	0	0	WLAN TX goes through
1	1	0	Stop WLAN TX

## 20.5.2 TDM operations for coexistence

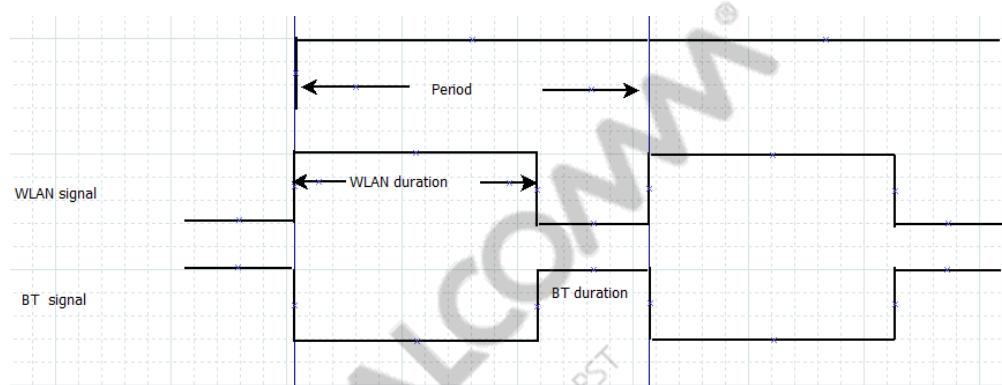
Certain BT profiles contain frequent BT traffic and it can always override low-priority WLAN traffic. To prevent this override, a certain time interval is offered for WLAN and BT traffic in a particular block or period of time. For example, in 100 ms, 80 ms can be offered for WLAN and 20 ms can be offered for BT to perform transmission. During the WLAN transmission period, all WLAN traffic is marked as high priority so that low-priority BT traffic does not override low-priority WLAN traffic in the WLAN transmission period. This methodology ensures WLAN low-priority traffic is not impacted when frequent low-priority BT traffic is present. COEX firmware

module processes two parameters— time period for which WLAN Tx must be active and the total duration allotted for WLAN and BT traffic.

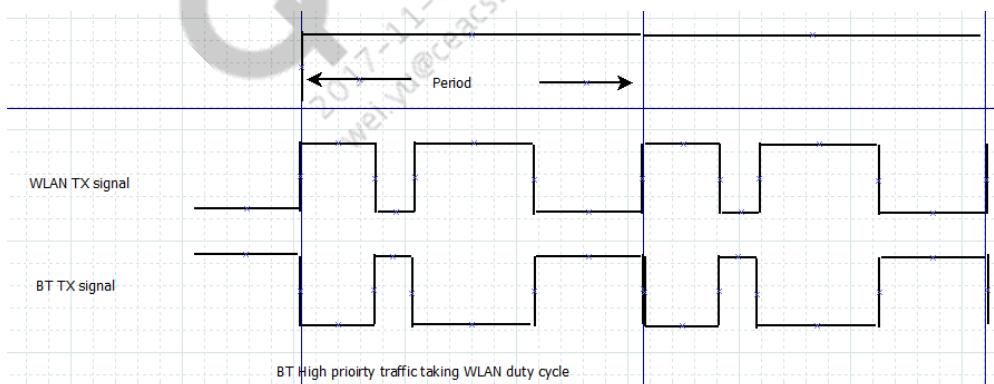
$$\text{WLAN TX Duty cycle} = \text{Time WLAN TX is active} / \text{Period} \quad (1)$$

The following timing diagrams describe different scenarios:

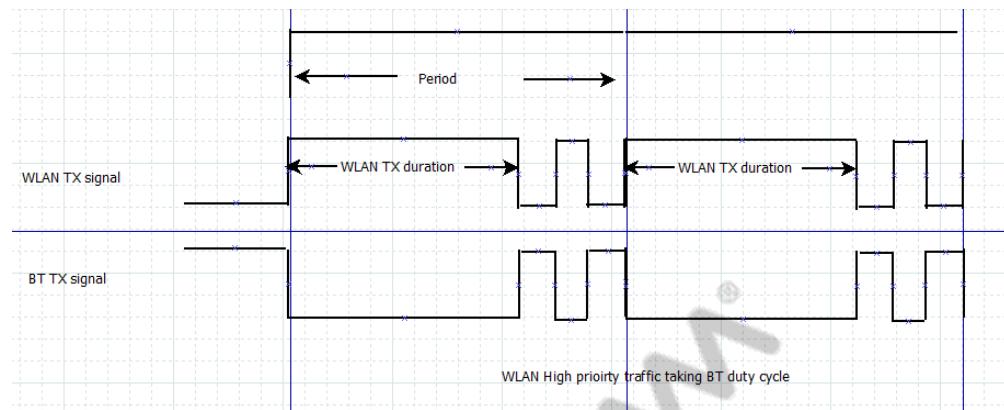
**Scenario 1: BT does not interfere in WLAN duration and WLAN does not interfere in BT duration**



**Scenario 2: BT high-priority traffic overrides WLAN transmission in WLAN period**



**Scenario 3: WLAN high-priority traffic overrides BT low-priority traffic in BT period**



### 20.5.3 Baseline priority table for COEX

The following table lists the baseline priority values for coexistence (COEX) with duty cycle. By default, voice, management, and beacon WLAN traffic are of high priority and this WLAN priority can be configured from host.

**Table 20-1 Priority table for BT and WLAN**

Wi-Fi Traffic Class	BT Priority	Duty cycle	[0 – Block BT, 1 – Unblock BT]	WLAN_PRIORITY GPIO
LOW	HIGH	WLAN	1	1
HIGH	HIGH	BT	1	0
HIGH	LOW	WLAN	0	0
LOW	LOW	WLAN	0	0
HIGH	LOW	BT	0	0
LOW	LOW	BT	1	1

### 20.5.4 Configure BT coexistence

To configure BT coexistence, do the following:

1. Set WLAN traffic priority by entering the following command:

```
iwpriv wifi0 btcoex_wl_pri 0x34
```

The following are WLAN priority bits and a bit set indicates particular WLAN traffic is of high priority.

Best effort—Bit 0, Background—Bit 1, Video—Bit 2, Voice—Bit 3, Beacon—Bit 4, and Management—Bit 5. Therefore, 0x34 = 110100 implies Video, Beacon and management traffic are high priority.

2. Display current priority by entering the following command:

```
iwpriv wifi0 g_btcoex_wl_pri
```

3. Enable BT coex feature by entering the following command:

- ```
iwpriv wifi0 btcoex_enable 1
```
4. Disable BT coex feature status :

```
iwpriv wifi0 btcoex_enable 0
```

  5. Display current status by entering the following command:

```
iwpriv wifi0 g_btcoex_enable
```

  6. Display current duty cycle parameter by entering the following command:

```
iwpriv wifi0 g_btcoex_dc
```

  7. Set duty cycle parameters by entering the following command:  
*iwpriv wifi0 btcoex\_dc <period> <wlan\_duration>*  
For example, enter *iwpriv wifi0 btcoex\_dc 100 80*

## 20.5.5 Configure BT on the two devices

To connect from device 1, enter the following commands:

```
export BTHOST_8311_SOC_TYPE=sdio
export BTHOST_BD_ADDR="0x00025B98A187"
./LinuxSPPLE 2 /dev/ttyHS0 115200
SPPLE>RegisterSPPLE
SPPLE> AdvertiseLE 1
SPPLE> StartScanning
```

To connect from device 2, enter the following commands:

```
export BTHOST_8311_SOC_TYPE=sdio
./LinuxSPPLE 2 /dev/ttyHS0 115200
SPPLE> connectLE 00025B98A187 0
SPPLE>send 100 - to send 100 bytes
```

## 20.5.6 Load firmware image

To load the firmware image and restart WLAN, do the following:

1. Enter the following command to copy the firmware images, ath wlan.bin and ath wlan.codeswap.bin, from backbone Linux PC to the appropriate location on the board:  
*scp ath wlan\*.bin root@192.168.1.1:/lib/firmware/QCA9984/hw.1*
2. Enable 2G WLAN in /etc/config/wireless by changing option disabled to 1 as follows:

```
config wifi-device wifi0
    option type qcawifi
    option channel auto
    option macaddr 8c:fd:f0:06:bc:cb
    option hwmode 11ng
    # REMOVE THIS LINE TO ENABLE WIFI:
    option disabled 1

config wifi-iface
    option device wifi0
```

```

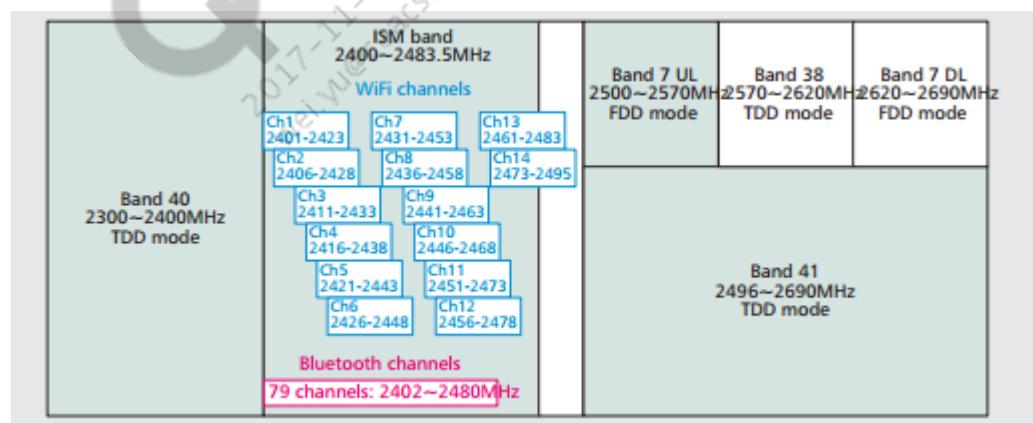
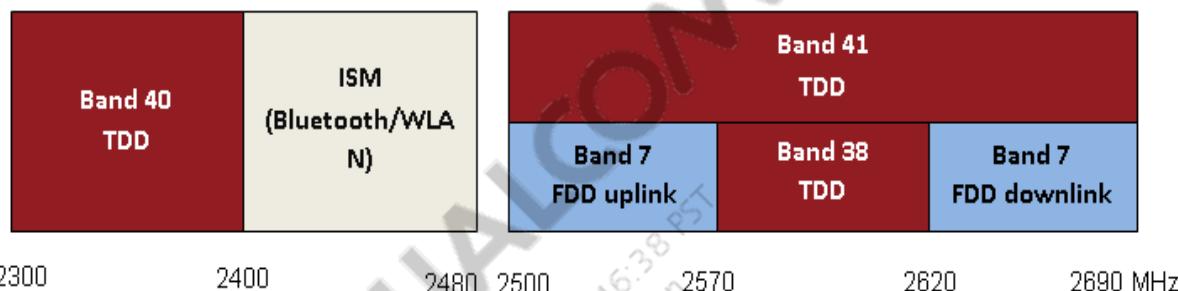
option network lan
option mode ap
option ssid Sample-ap
option encryption none

```

3. Restart WLAN to load new firmware by entering the wifi command.

## 20.6 LTE gateway and LTE coexistence

Wireless Local Area Network (WLAN) and Bluetooth share the same ISM band. LTE shares adjacent bands (band 40 /41/38 for TDD, band 7 for FDD) with WLAN and can have severe interference due to adjacent bands and limited isolation.



DK06 with WLAN 2G chipset on the platform. Sierra Wireless card is also present on DK06 through PCIe. This is the LTE modem on the platform. LTE Band 40 and Band 7 will affect most of the WLAN channels. There will be scenarios where LTE and WLAN will be brought up in the same channel also.

Therefore, it is necessary to have LTE Co-Ex where the WLAN has to operate and live with the desensitization. This can be done using LTE Channel avoidance as follows:

- WLAN firmware is informed using LTE modem – Sierra / MDM ) through WLAN host on the LTE band currently being used.

- Through ACS, WLAN channel is moved far away from the LTE Band ( as per implementation )
  - If LTE Freq < 2360MHz, WLAN to move to Channel 6 or 11 ( LTE in TDD )
  - If LTE Freq 2360 – 2380 , WLAN to move to Channel 11 ( LTE in TDD )
  - If LTE Freq <= 2510 or LTE Freq > 2510, WLAN to move to Channel 1 ( LTE in FDD )

This section applies only for LTE customers. Sierra-cm is LTE based connection manager where we used Sierra's EM7455 wireless module on top of Qualcomm's Dakota platform based router (DK06). This connection manager is supported as 2 packages (ipk). The AP.DK06 board supports LTE connections using the connection manager called sierra-cm. The Sierra EM7455 module is referenced here, and uses the PCIe port.

1. After installing the connection manager, the following config file can be viewed.

```
root@OpenWrt:/# cat /etc/config/sierra-cm
config sierra-cm 'config'
  option disabled '1'

  config profile
    option name 'default'
    option enabled '1'

  # User can create own profiles using below values to start LTE
  # connection manager.
  #
  #           name : user can decide any name
  #           enabled : 0 - inactive, 1 - active
  #                           (any 1 profile can be active at a time)
  #           connectiontype : LTE
  #
  #           ipfamily : 4 - IPv4, 6 - IPv6, 7 - IPv4v6
  #           pdptype : 0 - IPv4, 1 - PPP, 2 - IPV6, 3 - IPV4V6
  #
  #           ipaddress : xx.xx.xx.xx
  #           primarydns : xx.xx.xx.xx
  #           secondarydns : xx.xx.xx.xx
  #
  #                           (above format is for IPv4, xx can be any number of
  # digits.
  #
  #           use proper format for IPv6)
  #           authvalue : 0 - None, 1 - PAP, 2 - CHAP, 3 - PAP/CHAP
  #
  #           apn : based on service provider
  #
  #           username : based on service provider
  #
  #           password : based on service provider

  config profile
    option name 'profile1'
    option enabled '0'
```

```

option connectiontype 'LTE'
option ipfamily '4'
option pdptype '0'
option ipaddress '0.0.0.0'
option primarydns '0.0.0.0'
option secondarydns '0.0.0.0'
option authvalue '0'
option apn 'airtelgprs.com'
option username ''
option password ''

```

2. Enable connection manager.

```

root@OpenWrt:/# uci set sierra-cm.config.disabled=0
root@OpenWrt:/# uci commit sierra-cm

```

3. Start connection manager.

```
root@OpenWrt:/# lte-cm start
```

4. After the connection starts successfully, bring up the eth2 interface using the network restart option.

```
root@OpenWrt:/# /etc/init.d/network restart
```

5. The eth2 interface is up with a valid IP address if the connection is established successfully

```
root@OpenWrt:/# ifconfig eth2
```

6. To stop the sierra cm, run the following command:

```
root@OpenWrt:/# lte-cm stop
```

7. Access the generated logs for sierra connection manager either of the following ways:

- a. logread output

```
root@OpenWrt:/# logread
```

- b. sierracm file

```
root@OpenWrt:/# cat /var/logs/sierracm
```

8. Set up the APN.

APN settings are configured in a manner similar to profiles in config file. The default profile is enabled. Users can add their own profiles by referring the sample profile named **profile1** and enable it by setting the enabled value as 1. Make sure that the enabled value in the default profile is set to 0 in this case. The valid options and their values for profiles are provided in config file in the comments section.

## 20.6.1 LTE gateway configuration GUI

The LTE GUI is presented as a web page. Use a browser to navigate to the default address of <http://192.168.X.X> (where X.X is dependent on the implementation of the network) and enter the authentication credentials to log in to the GUI. Click the Network link on the left pane to expand

and view the options that are available under the Network section. Click the LTE link under the Network section to configure LTE gateway settings.

The LTE page is displayed, with a tabular layout of all of the previously configured profiles.

To configure LTE gateway settings, do the following:

1. Select the Enable LTE Connectivity check box to enable and start the connection manager. Sierra-cm is LTE based connection manager that uses Sierra's EM7455 wireless module on top of Qualcomm's IPQ4019 platform-based router (DK06). This connection manager is supported as 2 packages (ipk). The AP.DK06 board supports LTE connections using the connection manager called sierra-cm. The Sierra EM7455 module is referenced here, and uses the PCIe port.
2. Click Add Profile to create a new profile. A row is added under the LTE Network Profiles section and the page is refreshed.
3. Specify the following parameters for the LTE profile:
  - a. The Row column indicates the serial number of the LTE profile added on the page.
  - b. In the Active column, select Active from the shortcut menu that displays when you click inside the cell to enable the profile. Select Inactive to disable the profile. APN settings are configured using the LTE page in a manner similar to profiles in config files. The default profile is enabled. Users can add their own profiles by referring the sample profile named profile1 and enable it by setting the enabled value as 1.
  - c. In the Profile column, enter an easily-identifiable, meaningful name of the LTE profile that is being added.
  - d. In the PDP column, enter the packet data protocol (PDP) type. 0 signifies IPv4, 1 indicates PPP, 2 indicates IPv6, and 3 represents IPv4/v6.
  - e. In the Address column, enter the IP address of the gateway to be used for the LTE profile in dotted decimal format.
  - f. In the Primary DNS and Secondary DNS columns, enter the IP addresses of the primary and secondary domain name servers (DNS) to be used for address resolution. Enter the IPv4 or IPv6 address of the DNS server, based on your network deployment.
  - g. In the Auth value column, enter the authentication protocol to be used. 0 indicates no authentication, 1 represents PAP, 2 denotes CHAP, and 3 signifies PAP/CHAP.
  - h. In the APN Name column, enter the name of the access point node to be used, based on the service provider.
  - i. In the Username column, enter the name of the user that can log in to the APN, based on the service provider.
4. Click 'Save & Apply' to submit the configuration and for the changes to become effective for LTE profiles.
5. Click Reset to reload the page and disregard changes.

# 21 WiGig (802.11ad Standard) Features

---

This chapter describes the IEEE 802.11ad specification-compatible functionalities, which enable operations on 60 GHz frequency bands.

## 21.1 802.11ad WMI commands for management packets and antenna control

**NOTE** For complete information regarding the WMI commands available for 802.11ad configuration, refer to the *Qualcomm 60 GHz SDK User Guide* (80-YA400-5)

### 21.1.1 Management packets retry limit

The following WMI commands are used to configure the management retry limit:

#### **WMI\_SET\_MGMT\_RETRY\_LIMIT\_CMD**

The driver uses this command to set limit of retry to be used for sending management packets.

| Name              | Field Size [bytes] | Description                                                 |
|-------------------|--------------------|-------------------------------------------------------------|
| mgmt._retry_limit | 1 unsigned         | MAC retransmission for management frames (in range [1,254]) |
| Reserved          | 3 unsigned         | align to 4bytes boundary – this field shall be ignored      |

#### **WMI\_SET\_MGMT\_RETRY\_LIMIT\_EVENT**

The firmware sends this event to the driver in response to WMI\_SET\_MGMT\_RETRY\_LIMIT\_CMD request. The event provides success/fail status of the requested command.

| Name     | Field Size [bytes] | Description                                                    |
|----------|--------------------|----------------------------------------------------------------|
| Status   | 1 unsigned         | Result status code:<br>0x0 SUCCESS<br>0x1 BAD_PARAMETERS_ERROR |
| Reserved | 3 unsigned         | align to 4bytes boundary – this field shall be ignored         |

### **WMI\_GET\_MGMT\_RETRY\_LIMIT\_CMD**

The driver uses this command to retrieve content from FW on the currently used limit for MAC retransmission of management frames. Command with empty payload.

### **WMI\_GET\_MGMT\_RETRY\_LIMIT\_EVENT**

The firmware sends this event to the driver in response to WMI\_GET\_MGMT\_RETRY\_LIMIT\_CMD request. The event provides the currently used limit for MAC retransmission management frames.

| Name              | Field Size [bytes] | Description                                                                |
|-------------------|--------------------|----------------------------------------------------------------------------|
| mgmt._retry_limit | 1 unsigned         | Currently used MAC retransmission for management frames (in range [1,254]) |
| Reserved          | 3 unsigned         | align to 4bytes boundary – this field shall be ignored                     |

## **21.1.2 Antenna control**

The following WMI commands enable the configuration of the antenna control functionality:

### **WMI\_SET\_RF\_SECTOR\_PARAMS\_CMD**

The driver uses this command to set forced content to specified RF RX/TX sector of one or more RF chips. The new sector content is written to sectors table located in internal memory of destination RF modules specified by <rf\_modules\_vec>. Modified sector can be actually activated by sending additional command WMI\_SET\_SELECTED\_RF\_SECTOR\_INDEX\_CMD (or might be activated spontaneously if its index is selected by BRP/TXSS algorithms).

Power Management is disabled prior to sending command request. Otherwise, the modified sector content might be spontaneously overridden by restoring RF chip settings on exit from power sleep. Only a single command request at a time is allowed; that is, the host driver/user does not send a new request until the completion event for the previous command is received.

| Name        | Field Size [bytes] | Description                                                                |
|-------------|--------------------|----------------------------------------------------------------------------|
| sector_idx  | 2 unsigned         | Index of the RF sector which content should be modified (in range [0,127]) |
| sector_type | 1 unsigned         | Type of requested RF sector<br>RX = 0<br>TX = 1                            |

| Name               | Field Size [bytes]  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rf_modules_vec     | 1 unsigned          | Bitmask vector (8-bits) specifying destination RF modules<br>For example: rf_modules_vec=0x05 specifies destination modules RFC#0 , RFC#2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| rf_sectors_info[8] | 192 = 8x24 unsigned | Array of structures with entry#k filled by new content of RF sector to be forced in RF module #k. Entries corresponding to RF modules not selected in <rf_modules_vec> shall be ignored by FW.<br>Each entry in the rf_sectors_info[8] array contains six fields (4bytes each) comprising content of the RF sector:<br>rf_sectors_info array entry structure:<br>1) psh_hi (4bytes) - phase <PSH_H> field of the sector<br>2) psh_lo (4bytes) - phase <PSH_L> field of the sector<br>3) etype0 (4bytes) – bit0 of the edge amplifier gain index written to <ETYPE0> field of the sector<br>4) etype1 (4bytes) - bit1 of the edge amplifier gain index written to <ETYPE1> field of the sector<br>5) etype2 (4bytes) - bit2 of the edge amplifier gain index written to <ETYPE> field of the sector<br>6) dtype_swch_off_val (4bytes) - gain index of the Dist amplifier written to <DTYPE_SWCH_OFF> of the sector |

### **WMI\_SET\_RF\_SECTOR\_PARAMS\_DONE\_EVENT**

The firmware sends this event to the driver in response to WMI\_SET\_RF\_SECTOR\_PARAMS\_CMD request. The event provides success/fail status of the requested command.

| Name     | Field Size [bytes] | Description                                                                                                                                               |
|----------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| status   | 1 unsigned         | Result status code:<br>0x0 SUCCESS<br>0x1 BAD_PARAMETERS_ERROR<br>0x2 BUSY_ERROR (previous cmd request is still not completed)<br>0x3 NOT_SUPPORTED_ERROR |
| reserved | 3 unsigned         | align to 4bytes boundary – this field shall be ignored                                                                                                    |

### **WMI\_SET\_SELECTED\_RF\_SECTOR\_INDEX\_CMD**

The driver uses this command to force value of RF sector index currently selected by TXSS (TX sector) or BRP (RX sector) algorithms for communication with specified station.

For RX sector index selection, BRP algorithm should be disabled prior to the command execution, otherwise forced sector index might be overridden by spontaneous BRP execution.

TX sector index can be modified without disabling TXSS algorithm (forced value overrides results of TXSS algorithm).

Forcing special value section\_idx=0xFFFF unlocks active TX sector index, so it returns to normal behavior; that is, being updated by TXSS flow. Only single command request at a time is allowed;

that is, the host driver/user does not send new request until completion event for previous command is received.

| Name        | Field Size [bytes] | Description                                                                                                                                                     |
|-------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| cid         | 1 unsigned         | Connection/Station ID in [0:7] range                                                                                                                            |
| sector_type | 1 unsigned         | Type of requested RF sector<br>RX = 0<br>TX = 1                                                                                                                 |
| sector_idx  | 2 unsigned         | Forced sector index overriding BRP (RX) or TXSS (TX) algorithms<br>Special value=0xFFFF unlocks active sector index and returns it back to normal update policy |

### **WMI\_SET\_SELECTED\_RF\_SECTOR\_INDEX\_DONE\_EVENT**

The firmware sends this event to the driver in response to WMI\_SET\_SELECTED\_RF\_SECTOR\_INDEX\_CMD request. The event provides success/fail status of the requested command.

| Name     | Field Size [bytes] | Description                                                                                                                                                   |
|----------|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| status   | 1 unsigned         | Result status code:<br>0x0 SUCCESS<br>0x1 BAD_PARAMETERS_ERROR<br>0x2 BUSY_ERROR (previous command request is still not completed)<br>0x3 NOT_SUPPORTED_ERROR |
| reserved | 3 unsigned         | Align to 4bytes boundary                                                                                                                                      |

### **WMI\_SET\_RF\_SECTOR\_ON\_CMD**

**NOTE** This command is only supported in test mode. The command is not supported in operational mode.

The command switches RF logic to requested TX/RX mode and applies content of specified sector to RF hardware.

| Name           | Field Size [bytes] | Description                                     |
|----------------|--------------------|-------------------------------------------------|
| sector_idx     | 1 unsigned         | Sector index to be activated in [0:127] range   |
| sector_type    | 1 unsigned         | Type of requested RF sector<br>RX = 0<br>TX = 1 |
| rf_modules_vec | 2 unsigned         | bitmask vector specifying destination RF module |

### **WMI\_SET\_RF\_SECTOR\_ON\_DONE\_EVENT**

The firmware sends this event in response to WMI\_SET\_RF\_SECTOR\_ON\_CMD request. The event provides success/fail status of the requested command.

| Name     | Field Size [bytes] | Description                                                                                                                                               |
|----------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| status   | 1 unsigned         | Result status code:<br>0x0 SUCCESS<br>0x1 BAD_PARAMETERS_ERROR<br>0x2 BUSY_ERROR (previous cmd request is still not completed)<br>0x3 NOT_SUPPORTED_ERROR |
| reserved | 3 unsigned         | align to 4bytes boundary                                                                                                                                  |

### **21.1.3 Antenna state readback for 802.11ad**

The following WMI commands enable the configuration of the antenna state readback functionality:

#### **WMI\_GET\_RF\_SECTOR\_PARAMS\_CMD**

The driver uses this command to retrieve content of specified RF RX/TX sector from the internal memory of RF chip(s) specified by rf\_modules\_vec.

The firmware responds to reception of this command by sending WMI\_GET\_RF\_SECTOR\_PARAMS\_DONE\_EVENT with sector content information retrieved according to the request.

Only a single command request at a time is allowed; that is, host driver/user shall not send new request until completion event for previous command is received.

| Name           | Field Size [bytes] | Description                                                                                                                               |
|----------------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| sector_idx     | 2 unsigned         | Index of RF sector to be retrieved (in range [0,127])                                                                                     |
| sector_type    | 1 unsigned         | Type of requested RF sector<br>RX = 0<br>TX = 1                                                                                           |
| rf_modules_vec | 1 unsigned         | Bitmask vector (8-bits) specifying destination RF modules<br>For example: rf_modules_vec=0x05 specifies destination modules RFC#0 , RFC#2 |

#### **WMI\_GET\_RF\_SECTOR\_PARAMS\_DONE\_EVENT**

The firmware sends this event to the driver in response to WMI\_GET\_RF\_SECTOR\_PARAMS\_CMD request. The event provides success/fail status and contains sector information retrieved from internal memory of the RF chip(s) according to command request.

| Name               | Field Size [bytes]     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| status             | 1 unsigned             | Result status code:<br>0x0 SUCCESS<br>0x1 BAD_PARAMETERS_ERROR<br>0x2 BUSY_ERROR (previous cmd request is still not completed)<br>0x3 NOT_SUPPORTED_ERROR                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| reserved           | 7 unsigned             | Align next field to 8bytes boundary – this field shall be ignored                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| tsf                | 8 unsigned             | TSF timestamp when RF sectors where retrieved                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| rf_sectors_info[8] | 192 = 8x24<br>unsigned | Array of structures with entry#k filled by content of<br>RF sector retrieved from RF module #k (as requested in <rf_modules_vec> of the original WMI_GET_RF_SECTOR_PARAMS_CMD).<br>Entries corresponding to RF modules not selected in <rf_modules_vec> shall be ignored (will be filled with zeros)<br>Each entry in the returned rf_sectors_info[8] array contains six fields (4bytes each) comprising content of the RF sector:<br><b>rf_sectors_info array entry structure:</b><br>1) psh_hi (4bytes) - phase <PSH_H> field retrieved from the sector<br>2) psh_lo (4bytes) - phase <PSH_L> field retrieved from the sector<br>3) etype0 (4bytes) - bit0 of the edge amplifier gain index retrieved from <ETYPE0> field of the sector<br>4) etype1 (4bytes) - bit1 of the edge amplifier gain index retrieved from <ETYPE1> field of the sector<br>5) etype2 (4bytes) - bit2 of the edge amplifier gain index retrieved from <ETYPE> field of the sector<br>6) dtype_swch_off_val (4bytes) - gain index of the Dist amplifier as retrieved from <DTYPE_SWCH_OFF> of the sector |

### WMI\_GET\_SELECTED\_RF\_SECTOR\_INDEX\_CMD

The driver uses this command to retrieve RF sector index currently selected by TXSS/BRP algorithm for communication with specified station.

Only a single command request at a time is allowed; that is, host driver/user shall not send new request until completion event for previous command is received.

| Name        | Field Size [bytes] | Description                                     |
|-------------|--------------------|-------------------------------------------------|
| cid         | 1 unsigned         | Connection/Station ID in [0:7] range            |
| sector_type | 1 unsigned         | Type of requested RF sector<br>RX = 0<br>TX = 1 |
| reserved    | 2 unsigned         | align to 4bytes boundary                        |

### WMI\_GET\_SELECTED\_RF\_SECTOR\_INDEX\_DONE\_EVENT

The firmware sends this event to the driver in response to WMI\_GET\_SELECTED\_RF\_SECTOR\_INDEX\_CMD request. The event provides success/fail status of the requested

command along with RF sector index currently selected by TXSS/BRP algorithm for communication with specified station.

| Name       | Field Size [bytes] | Description                                                                                                                                               |
|------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| sector_idx | 2 unsigned         | Retrieved sector index selected in TXSS (for TX sector request) or BRP (for RX sector request)                                                            |
| status     | 1 unsigned         | Result status code:<br>0x0 SUCCESS<br>0x1 BAD_PARAMETERS_ERROR<br>0x2 BUSY_ERROR (previous cmd request is still not completed)<br>0x3 NOT_SUPPORTED_ERROR |
| reserved   | 5 unsigned         | align next field to 8bytes boundary                                                                                                                       |
| tsf        | 8 unsigned         | TSF timestamp when RF sectors index was                                                                                                                   |

## 21.1.4 WMI command IDs

The following table lists the new WMI commands and IDs to the device.

| Command name                           | ID     |
|----------------------------------------|--------|
| WMI_SET_MGMT_RETRY_LIMIT_CMDID         | 0x0930 |
| WMI_GET_MGMT_RETRY_LIMIT_CMDID         | 0x0931 |
| WMI_GET_RF_SECTOR_PARAMS_CMDID         | 0x09A0 |
| WMI_SET_RF_SECTOR_PARAMS_CMDID         | 0x09A1 |
| WMI_GET_SELECTED_RF_SECTOR_INDEX_CMDID | 0x09A2 |
| WMI_SET_SELECTED_RF_SECTOR_INDEX_CMDID | 0x09A3 |
| WMI_SET_RF_SECTOR_ON_CMDID             | 0x09A4 |

### 21.1.5 WMI event IDs

The following table describes the list of new WMI events.

| Event name                                    | ID     |
|-----------------------------------------------|--------|
| WMI_SET_MGMT_RETRY_LIMIT_EVENTID              | 0x1930 |
| WMI_GET_MGMT_RETRY_LIMIT_EVENTID              | 0x1931 |
| WMI_GET_RF_SECTOR_PARAMS_DONE_EVENTID         | 0x19A0 |
| WMI_SET_RF_SECTOR_PARAMS_DONE_EVENTID         | 0x19A1 |
| WMI_GET_SELECTED_RF_SECTOR_INDEX_DONE_EVENTI  | 0x19A2 |
| WMI_SET_SELECTED_RF_SECTOR_INDEX_DONE_EVENTID | 0x19A3 |
| WMI_SET_RF_SECTOR_ON_DONE_EVENTID             | 0x19A4 |

## 21.2 802.11ad firmware recovery

This section describes the 802.11ad firmware crash recovery architecture for Qualcomm® Internet Processor (IPQ) targets. It also describes the tasks of host software and firmware to support this feature.

The main goal of this feature is to reinitialize the host and reload the firmware to its initial state at the event of firmware crash.

The following table lists the definitions of terms and abbreviations:

**Table 21-1 Acronyms**

| Acronym  | Definition                                                                                                                                         |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| Host     | Application processor environment where the applications run. For PC/Mac OS platforms, it is the main processor executing the OS and applications. |
| Target   | Dedicated CPU as part of the hardware SoC solution provided by Qualcomm.                                                                           |
| Firmware | Software running on the target CPU.                                                                                                                |
| Hardware | Refers to either MAC or CPU on the target.                                                                                                         |
| Software | Software running on the host CPU.                                                                                                                  |

When the firmware crashes:

- Host is interrupted by the target with a firmware event.
- Event handler on the host software identifies the event type and handles the event accordingly.
- Event handler schedules a work queue. The work queue either resets the target and reloads the firmware to the target in the firmware recovery mode, or asserts the host in the host assertion mode.
- The 802.11ad host driver detects firmware crash (assertion), and behaves in one of the two modes:

- Host assertion mode: Driver collects firmware crash dump at the event of firmware crash and panics the host, which results in reboot of the entire system.
- Firmware recovery mode: Driver resets the firmware and restarts it to a working state. All previous configurations are lost and it is up to the user-space to restore the AP configuration. This is the default mode. It is possible to set the host not to reset the firmware in this state.

## 21.2.1 Control commands

The configuration of the host behavior in the event of firmware assertions are done via module parameters. These may be set during insmod operation, or at runtime by writing to /sys/kernel/module/wil6210/parameters.

### 21.2.1.1 Host assertion control

The module parameter controls whether host gets asserted when a firmware assertion is detected. This results in rebooting of the entire system. The parameter name is `crash_on_fw_err` and is not set by default. It can be set:

- Either during the insmod operation:

```
insmod wil6210.ko crash_on_fw_err
```

- Or at runtime after the insmod operation:

```
echo 1 > /sys/module/wil6210/parameters/crash_on_fw_err
```

### 21.2.1.2 Firmware recovery mode

The firmware recovery mode is relevant only when the `crash_on_fw_err` parameter is not set. In such cases, the default behavior is to execute firmware recovery process by resetting the target, and reloading the firmware image to the target. It is also possible to disable the firmware recovery mode through a module parameter so that the host does not initiate recovery.

The module parameter to be set for this is `no_fw_recovery`. It may be set either during insmod operation or at runtime. For more details on this mode, see Q2.

Reconfiguration of the AP mode has to be done in the user-space; this is not a part of the feature.

## 21.3 802.11ad: Fast session transfer

Fast session transfer (FST) enables fast switching of a session from one band (channel) to another to allow better user mobility, dynamic channel conditions, and joint management of multiple bands. A multiband device can manage operation over more than one frequency band.

The 802.11ad specification defines transparent and nontransparent FST modes and allows both simultaneous and nonsimultaneous transfer. The specification requires:

- Multiband information: Elements added to some management frames such as beacon, probe request/response, and association request/response
- FST Setup Protocol using FST action frames

- When multiple connections exist, packets can transmit over a specific band or over different bands simultaneously

FST is not yet part of certification.

FST for IPQ8064-based releases with 802.11ad supports these features:

- Non-transparent FST (with option to use the same or different MAC addresses between bands)
- FST over two interfaces (802.11ad and 802.11ac)
- FST switch between different bands (switching between channels in the same band is not supported)
- STA-based switch (stream-based switch not supported)
- Association required in both bands before FST setup and switch
- Switch policy is well-defined and known to both STA and AP
- Switch back and forth to prevent STA/AP disagreement on band preference and FST situations  
Switch by explicit ACK Request/Response (switch by individually addressed MPDU not supported)

See the *Fast Session Transfer (FST) Manager for IPQ8064-Based Releases Overview* (80-Y9522-2) for more information about FST architecture and operations.

Because the FST settings are stored in the UCI wireless persistent database on the device, they are only required to be set once.

### 21.3.1 Configure the AP160 for 802.11ad

1. Start an 802.11ad AP in non-secured mode

```
uci set wireless.radio0.disabled=0
uci set wireless.@wifi-iface[0].encryption=none
uci commit wireless
/etc/init.d/network restart
```

2. Start an 802.11ad AP in secured mode

```
uci set wireless.radio0.disabled=0
uci set wireless.@wifi-iface[0].encryption=psk2+gcmp
uci set wireless.@wifi-iface[0].key="your_password"
uci commit wireless
/etc/init.d/network restart
```

3. Select the 802.11ad AP channel

```
uci set wireless.radio0.channel=2      (Supported channels are 1, 2, 3)
uci commit wireless
/etc/init.d/network restart
```

4. Start an 802.11ad AP in STA mode

```
uci set wireless.@wifi-iface[0].mode=sta
```

```
uci commit wireless
/etc/init.d/network restart
```

### 21.3.2 Connect an 802.11ad STA to an 802.11ad AP

To connect an AP160 802.11ad STA to an AP160 802.11ad AP:

1. Configure an AP160 as an 802.11ad AP.
2. Configure another AP160 as a STA.
3. On the 802.11ad STA:
  - a. Configure the IP address (or use **dhclient wlan0** for dynamic IP). Configure it to an address in the same subnet as the IP of the br-lan on the AP. This example uses **192.168.1.17**:  
`ifconfig wlan0 192.168.1.17`
  - b. Bring up the wlan0 interface:  
`ifconfig wlan0 up`
  - c. *For a non-secured connection*, use the iw tool to scan and connect:  
`iw wlan0 scan`

If this command returns a scan result with an *SSID=ap\_ssid\_name* that matches the AP160 AP, connect to this AP using the command:

```
iw wlan0 scan; iw wlan0 connect ap_ssid_name
```

To send packets to another AP160 AP:

Configure one of the AP160 APs to a different IP address than the default 192.168.1.1. The br-lan interface IP address must be different to avoid a conflict with the IP address of the remote br-lan interface. This example uses **192.168.1.100**.  
`ifconfig br-lan 192.168.1.100`

This can also be modified using configuration file. For example, to change the device IP address from the default 192.168.1.1 to **192.168.2.1**:

1. Change this line in the file **/etc/config/network**:

```
option ipaddr192.168.1.1
```

To:

```
option ipaddr192.168.2.1
```

2. Commit the settings by running this command:

```
/etc/init.d/network restart
```

**NOTE** This case requires users to relog in using SSH because the device is accessible at its new IP address.

### 21.3.3 Enter RF-kill state

```
uci set wireless.radio0.disabled=1
uci commit wireless
/etc/init.d/network restart
```

### 21.3.4 Enter automatic channel selection (ACS)

```
uci set wireless.radio0.channel=0
uci commit wireless
/etc/init.d/network restart
```

### 21.3.5 Enable Wi-Fi protected setup (WPS) functionality

```
uci set wireless.@wifi-iface[0].wps_config='push_button virtual_push_
button physical_push_button display virtual_display'
uci commit wireless
wifi
hostapd_cli -i wlan0 -p /var/run/hostapd-phy0 wps_pbc
```

After these commands, wps\_state will be configured in the hostapd conf file with 1 (open mode) or 2 (secure mode).

#### Use these steps to associate a STA:

1. Run a supplicant on the STA side.
2. Create a config file called **wpa\_config.cfg** with the content:  

```
ctrl_interface=/var/run/wpa_supplicant
```
3. Start the supplicant with the command:  

```
sudo wpa_supplicant -iwlan0 -Dnl80211 -c ./wpa_config.cfg -ddd
```
4. On another shell, open **wpa\_cli** (requires sudo otherwise will not connect to the supplicant):  

```
sudo wpa_cli
```
5. At the wpa\_cli > command prompt run a scan:  

```
> scan
OK
<3>CTRL-EVENT-SCAN-STARTED
<3>CTRL-EVENT-SCAN-RESULTS
<3>WPS-AP-AVAILABLE

> scan_results
bssid / frequency / signal level / flags / ssid
00:36:65:88:20:10      62640     80      [WPS][DMG][ESS]      wigig-
21-11ad
04:ce:14:00:05:42      60480     80      [WPS][DMG][ESS]      OpenWrt
```
6. Verify that WPS appears in the line for the AP160.
7. On the AP160, press the WPS button or run the **/etc/hotplug.d/button/50-wps** script.

8. On the STA wpa\_cli > command prompt, use this command to connect:

```
> wps_pbc
OK
<3>CTRL-EVENT-SCAN-STARTED
<3>CTRL-EVENT-SCAN-RESULTS
<3>WPS-AP-AVAILABLE-PBC
<3>Trying to associate with 04:ce:14:00:05:42 (SSID='OpenWrt'
freq=60480 MHz)
<3>Associated with 04:ce:14:00:05:42
<3>CTRL-EVENT-EAP-STARTED EAP authentication started
<3>CTRL-EVENT-EAP-STATUS status='started' parameter=
<3>CTRL-EVENT-EAP-PROPOSED-METHOD vendor=14122 method=1
<3>CTRL-EVENT-EAP-STATUS status='accept proposed method'
parameter='WSC'
<3>CTRL-EVENT-EAP-METHOD EAP vendor 14122 method 1 (WSC) selected
<3>WPS-CRED-RECEIVED
<3>WPS-SUCCESS
```

### 21.3.6 Enable fast session transfer (FST) functionality

To see the current FST configuration:

```
root@OpenWrt:/# uci show fst
fst.config=fst
fst.config.disabled='1'
fst.config.mux_interface='bond1'
fst.config.interface1='ath0'
fst.config.interface2='wlan0'
fst.config.interface1_priority='100'
fst.config.interface2_priority='110'
fst.config.interface2_mac=' '
```

The FST script is responsible for FST configuration and triggering FST manager and it is at: **/usr/sbin/fst.sh**

|                                   |              |  |  |
|-----------------------------------|--------------|--|--|
| <b>See current configuration:</b> | uci show fst |  |  |
|-----------------------------------|--------------|--|--|

|                                | uci set <parameter>=<new value>                                                                                                                                                                              |                    |  |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|--|
| Parameter                      | Description                                                                                                                                                                                                  | Default            |  |
| disabled                       | Specifies whether FST is enabled                                                                                                                                                                             | 0 (Disabled)       |  |
| mux_interface                  | Mux on top of the interface used as part of FST                                                                                                                                                              | bond1              |  |
| interface1                     | First interface used for FST (ath0 or ath1)                                                                                                                                                                  | ath0               |  |
| interface2                     | Second interface used for FST                                                                                                                                                                                | wlan0              |  |
| interface1_priority            | Priority of first interface; based on priority of interface, FST decides where to give the priority for data traffic                                                                                         | 100                |  |
| interface2_priority            | Priority of second interface; based on priority of interface, FST decides where to give the priority for data traffic                                                                                        | 110 (> interface1) |  |
| interface2_mac                 | MAC address of the second interface used for FST (optional parameter). Use this parameter to force setting both interfaces to the same MAC address. If not set, each interface will have its own MAC address |                    |  |
| <b>Save the configuration:</b> | uci commit fst<br>/etc/init.d/network restart                                                                                                                                                                |                    |  |

FST configurations can be seen and modified at: /etc/config/fst.

### 21.3.7 802.11ad: Board file configuration

By default, 11ad driver loads RF related board file with specific file name, wil6210.brd, which is located under /lib/firmware.

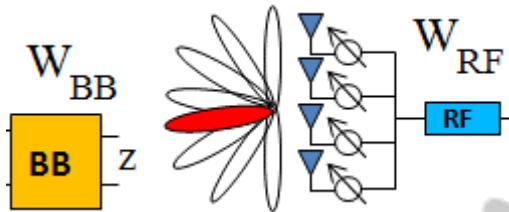
It is possible to configure an alternative board file name. This is useful for cases where more than one 11ad device is connected to the platform and for each a different board file is needed. Note that this configuration is unique for each device instance.

```
uci set wireless.radio0.board_file=my_board_file.brd
uci commit wireless
/etc/init.d/network restart
```

If multiple 11ad devices exist in the platform, the second device can be identified with radio1, which can be used in the aforementioned command instead of radio0.

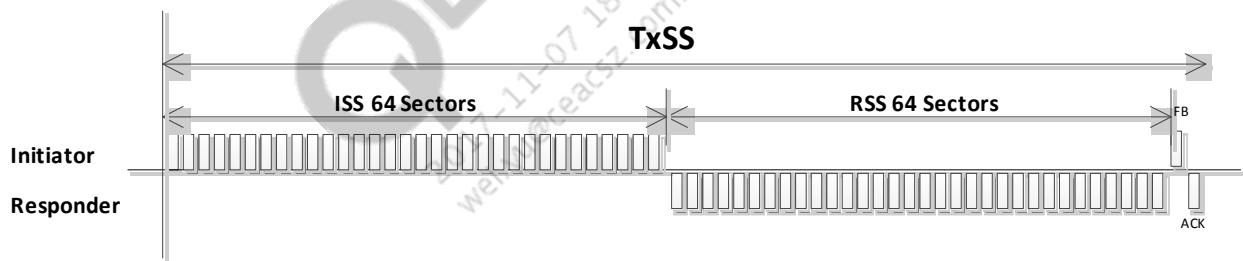
## 21.4 Massive array for 802.11ad

Massive or large array for WiGig is supported. In the massive array board file, Tx supports 126 directed equal EIRP Tx beam 40dBM EIRP , AZ: $\pm 45$  EL: $\pm 15$  (AZ: $\pm 60$  EL: $\pm 25$ ) and Rx supports omni-sector, low noise figure, and maximum gain. Massive array beamforming consists of extended SLS with 64 directed beams and massive array BRP. Massive Array calibrations consist of antenna gain (Tx, Rx), silence RSSI (noise calibration), and Rx signal gain (avoid saturation).



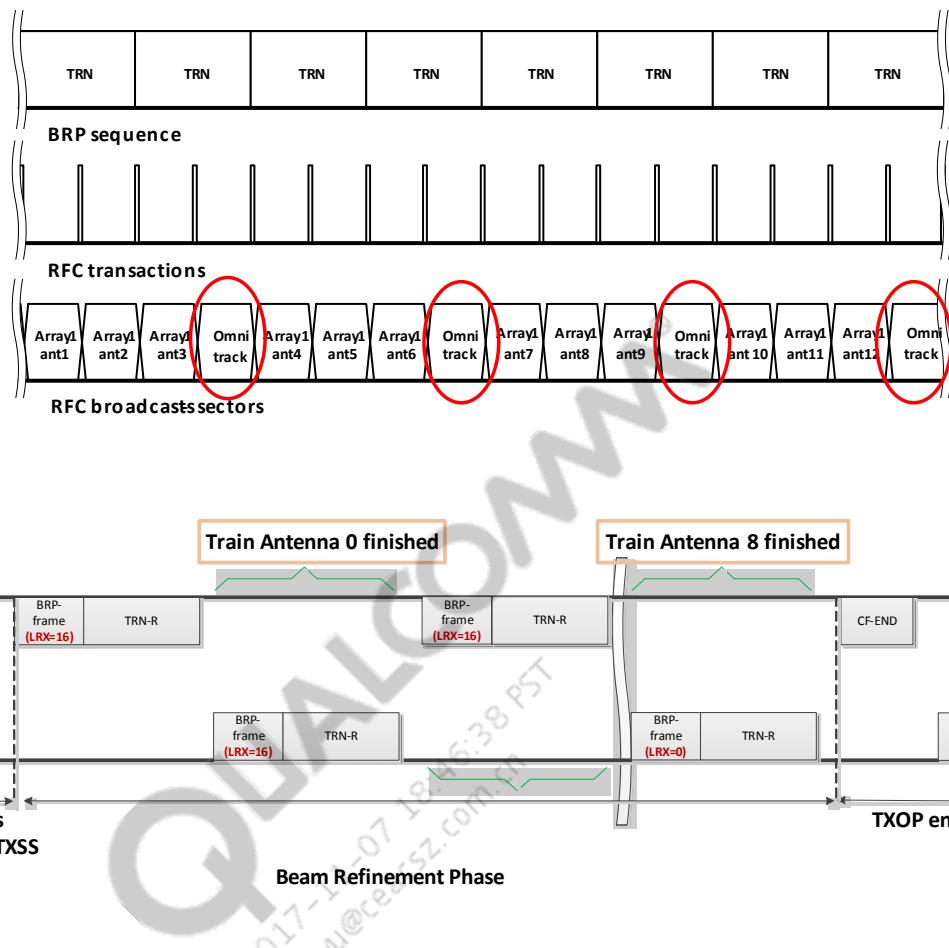
### QMA beamforming Tx sector sweep

Qualcomm Massive Array (QMA) uses 64 Tx vectors for incoming signal strength (ISS) and received signal strength (RSS). QMA is based on approximately 50-70 antennas from selected arrays. All sectors have same EIRP equal to 40 dB. Switching is done by a single RFC command (same as single array operation). The total time taken for Tx sector sweep is 2100  $\mu$ s.



### Symmetric flow in massive array BRP

Each Beam Refinement Protocol (BRP) frame trains a single array. In a single BRP frame, all single elements are measured. The reference is always the Omni sector. The number of BRP frames is equal to number of arrays (8 arrays). The BRP time for full Rx training is 1085  $\mu$ s. The total BF time is approximately 3200  $\mu$ s.



## 21.5 802.11ad system overview

Software development groups can integrate Qualcomm 60 GHz solution with OEM host driver. The system has two chips:

- Baseband (M) chip – Contains the baseband, PHY, MAC, DMA, PCIe, and so on
- RF (R) chip – Contains the RF antenna

The Baseband chip is assembled on a standard HMC/M2 and is connected to the standard Wi-Fi slot. It is connected to the QCA6310 (R chip) using a UFL cable.



Figure 21-1 Integrated HMC and RF module



Figure 21-2 HMC with RF and JTAG connected to auxiliary PCIe slot

The WiGig specification defines several use cases. This document provides the required background only to the Wi-Fi functionality (IP).

### Wi-Fi over 60 GHz (IP)

This is a basic networking flow in which OS transmits and receives packets over the 60 GHz air interface. The packets are passed by the driver from the OS to the hardware using a DMA.

- Station (Client) – an integrated system (for example, laptop) in which the Qualcomm 60 GHz device is managed by a driver. The driver is responsible for operating the chip via mailbox commands.
- AP (Access Point) – Integrated system with Qualcomm 60 GHz solution that runs AP services. The AP can be Tri-Band AP or Single band (60 GHz) AP. The driver is responsible for operating the chip via mailbox commands.

### 21.5.1 Wi-Fi system

In such a system, there are at least 2 devices where one is the client and the other is the PCP/AP. The same software stack is used to support different functionalities: Client or PCP/AP. Some of the basic supported functionalities are:

- BSS STA mode.
- PBSS STA mode
- PCP (PBSS)

The following figure represents 11ad use cases representing STA and AP in different configurations:



**Figure 21-3 Wi-Fi system examples**

## 21.5.2 Qualcomm 60 GHz device architecture

### 21.5.2.1 Hardware

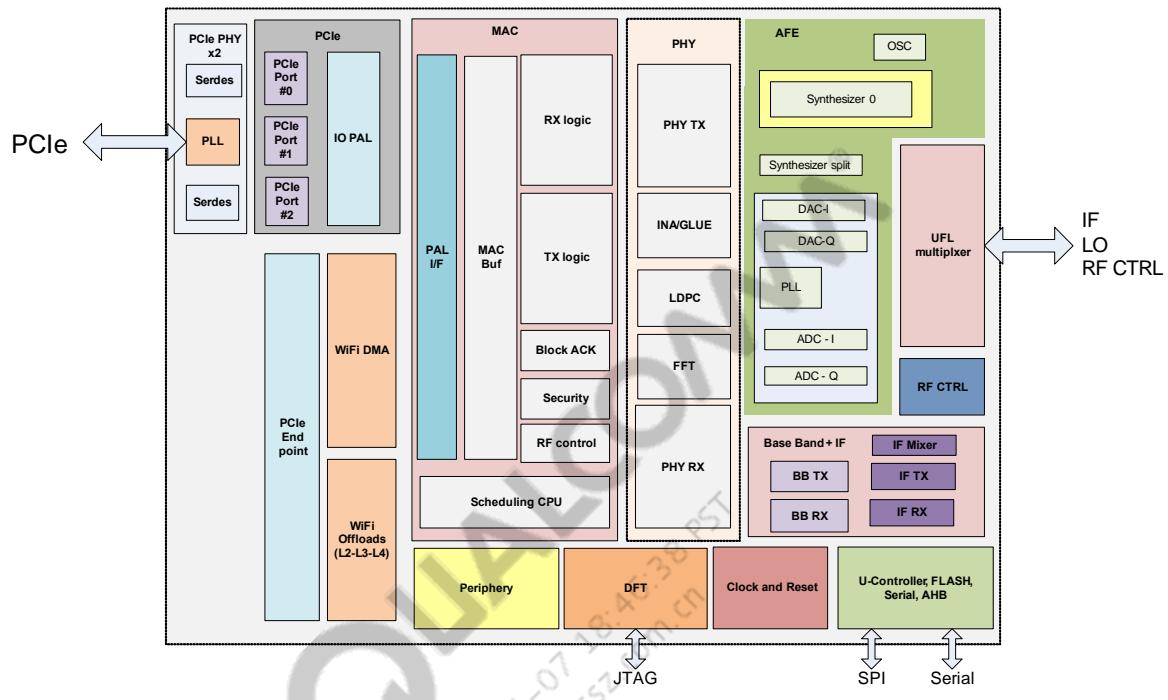


Figure 21-4 Qualcomm 60 GHz device high-level architecture

### 21.5.2.2 Software

Host – 802.11ad network (wireless) driver, system networking software stack

Device – Firmware & µCode runs on two CPUs (ARC)

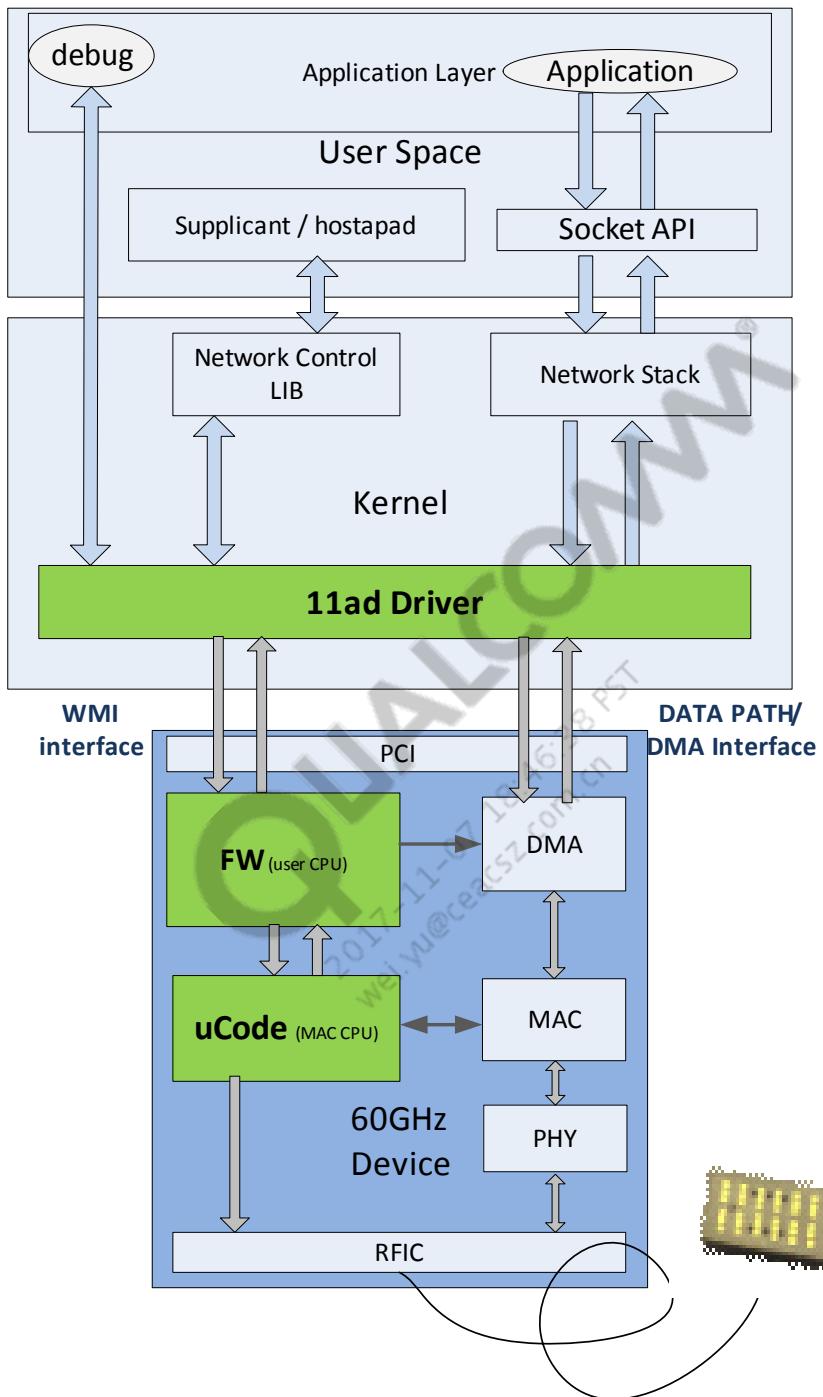


Figure 21-5 Software high-level architecture

### 21.5.3 Software partition

The following figure depicts the high-level software stack partition. The list of software elements and functionality follows.

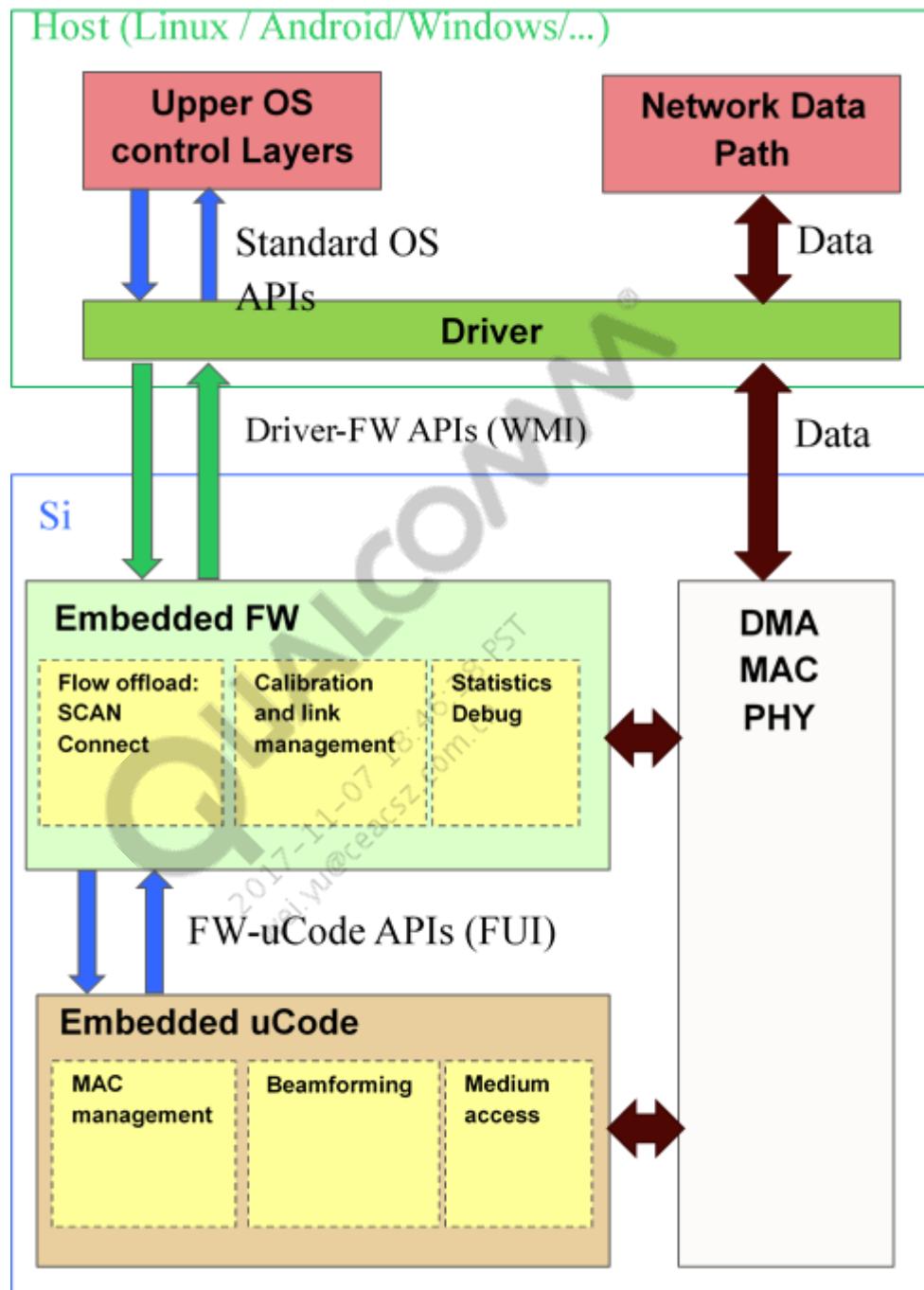


Figure 21-6 Windows software partition

### 21.5.3.1 Application

- Native OS network manager UI for interface network control.
- Implements API with driver for device access/control/debug.

### 21.5.3.2 Supplicant

- Trigger GUI events to driver layer
- Implement upper SME layer
- Implement WSC (and WPA2-PSK at first stage)

### 21.5.3.3 11ad driver

- Implement reset and firmware download flows
- Implement WMI (Wireless Module Interface) API for control and management
- Implement data path API, allocate, and manage resources
- Implement offloads

### 21.5.3.4 Firmware

Firmware is code that runs on the upper CPU of the Qualcomm 60 GHz device and is responsible for the following:

- Single thread context
- OS functions:
  - Scheduling, memory allocation, logging, error handling, and IPC
- Offloaded MAC architecture with various of MAC offloads
- Communication with driver (receive commands, send events)
  - Exposes two mailboxes – operational and debug
- Hardware configurations:
  - MAC queue management and configuration
  - DMA management and VRING switch flow
  - PCIe—Various flows and configurations (PCIe config space)
  - RF management—Loading of board file and configurations
  - OTP access—Read access; one-time write in production line
- Calibrations and temperature monitoring
  - Both baseband and RF chips
- MLME offloaded flows
  - Discovery manager
  - Legacy scan and P2P flows
  - Connection manager
  - Handle connection flows and maintain connections pool

- Link maintenance per connection.
- Maintain statistics per connection
- Link maintenance
  - Rate search enable and configurations, beamforming long-term trigger
- Power management
  - Power management logic tracks PCIe and MAC states and switches in and out of deep sleep.
- L2 manager is the entity that connects to the rest of the software modules and connects them into flows.
- Management of μCode (send commands, receive events)
- LOW SME is a basic translation layer from WMI commands into L2 manager functions.

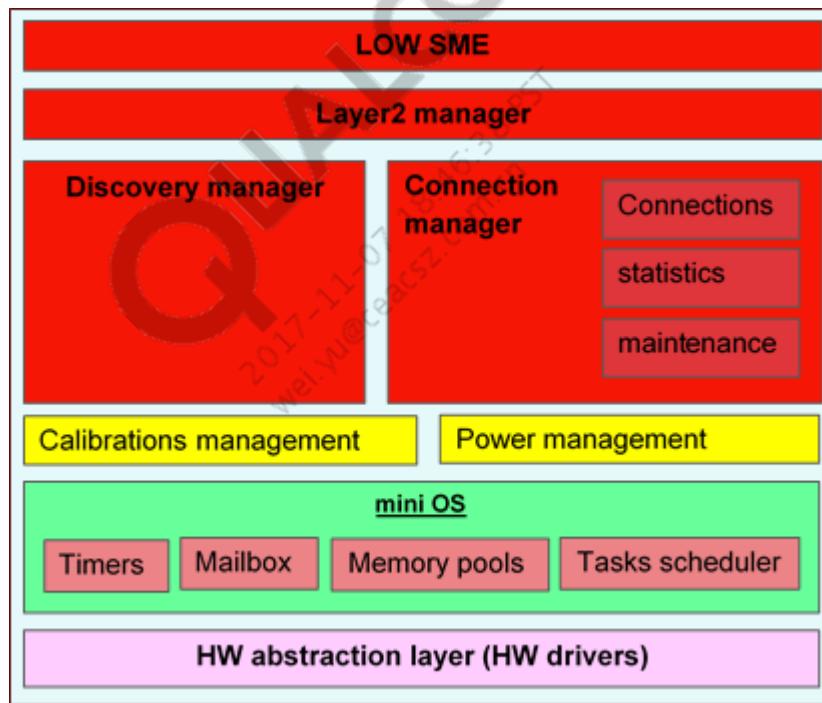


Figure 21-7 Firmware architecture

### 21.5.3.5 μCode

μcode (or microcode) runs on the lower CPU of the Qualcomm 60 GHz device. It is responsible for managing the PHY and MAC in a real time manner (sub millisecond reactions).

- The μCode is partitioned into three main layers:
- The lowest layer is the HAL for the MSXD. But there are also HALs for PHY/RF that are common to firmware.

- On top of the HALs there are the time critical flows as defined in spec. e.g. TxSS, BRP etc. The flows involve tight interaction between µCode and hardware.
- On top, there is the “software Modules layer” that is not as time sensitive—for example, power management.

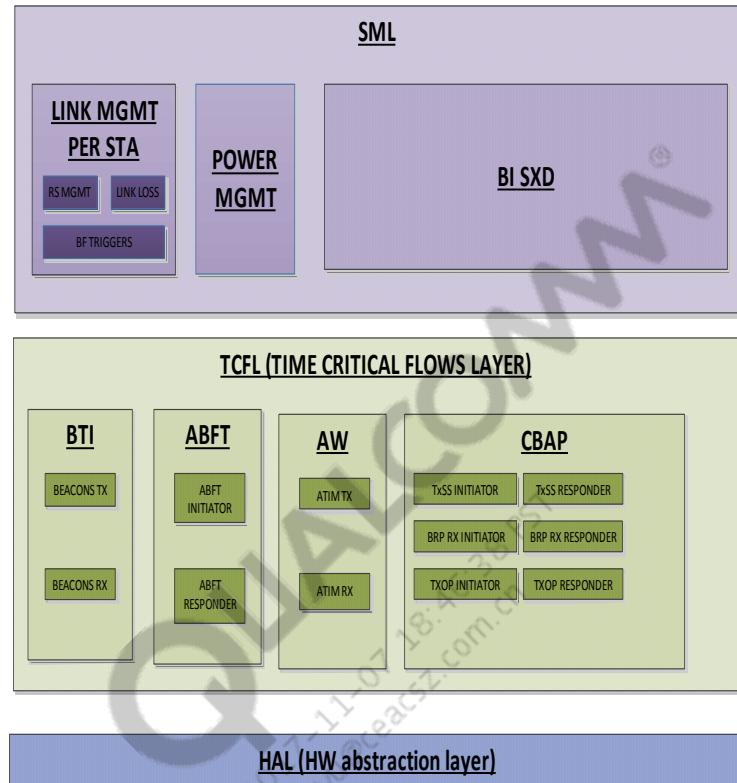


Figure 21-8 µCode architecture

## 21.6 Fast session transfer

Fast session transfer (FST) enables fast switching of a session from one band (channel) to another to allow better user mobility, dynamic channel conditions, and joint management of multiple bands. A multiband device can manage operation over more than one frequency band.

The 802.11ad specification defines transparent and nontransparent FST modes and allows both simultaneous and nonsimultaneous transfer. The specification requires:

- Multiband information: Elements added to some management frames such as beacon, probe request/response, and association request/response
- FST Setup Protocol using FST action frames
- FST Session Transfer: When multiple connections exist, packets can transmit over a specific band or over different bands simultaneously

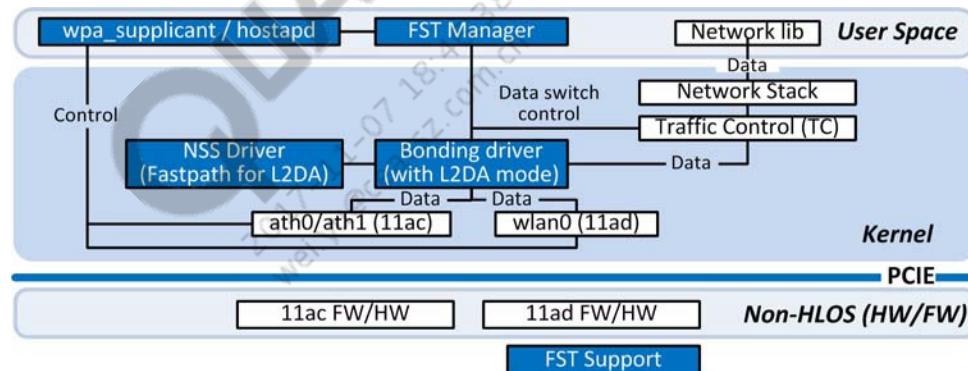
FST is not yet part of certification.

### FST for IPQ8064-based releases with 802.11ad supports these features:

- Non-transparent FST (with option to use the same or different MAC addresses between bands)
- FST over two interfaces (802.11ad and 802.11ac)
- FST switch between different bands (switching between channels in the same band is not supported)
- STA-based switch (stream-based switch not supported)
- Association required in both bands before FST setup and switch
- Switch policy is well-defined and known to both STA and AP
- Switch back and forth to prevent STA/AP disagreement on band preference and FST situations
- Switch by explicit ACK Request/Response (switch by individually addressed MPDU not supported)

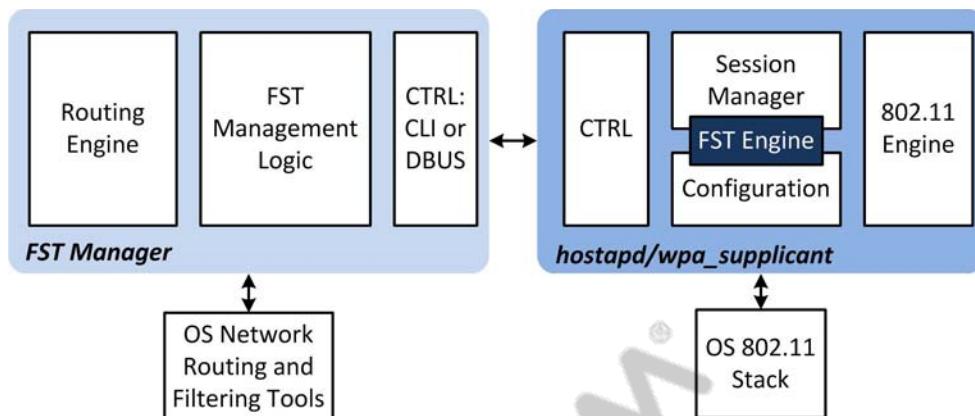
Refer to the *QCA9500 802.11ad 60-GHz UCI Commands User Guide* (80-Y9522-1) for more information on the FST UCI commands.

#### 21.6.1 FST architecture



**Figure 21-9 FST Architecture**

FST is implemented in two main paths—control path and data-path.



**Figure 21-10 FST Control Path and Data Path**

### 21.6.1.1 Control path

The control path identifies whether the connecting client is FST capable, and communicates with the client if it needs the FST switch (switch between 802.11ad and non-802.11ad interfaces for data traffic). The control path is implemented by the FST manager with the help of hostapd. Hostapd includes changes required for FST implementation. All Wi-Fi interfaces that are used for FST are managed by a single hostapd instance.

Hostapd contains an FST engine that implements the FST state machine as defined by the 802.11ad standard. The FST manager communicates with hostapd through its control interface.

### 21.6.1.2 Data path

In the data path, FST manager (through the routing engine) uses the bonding driver (an existing Linux driver) to bond multiple network interfaces. It allows exposing a single IP/MAC address to the network stack. The bonding driver may operate in different modes. The new L2DA mode was created to support FST and NSS fast path. The FST Wi-Fi interfaces are enslaved under the bonding driver, and data may pass through one or more interfaces. The underlying Wi-Fi drivers are mostly unaware of FST; their main requirement is to support FST action frame Tx/Rx and embed MB IEs in management frames.

FST manager configures the bonding driver interface to transmit a data packet based on its destination MAC address. A table listing MAC addresses and the interface to transmit the packet through it is maintained by the bonding L2DA module.

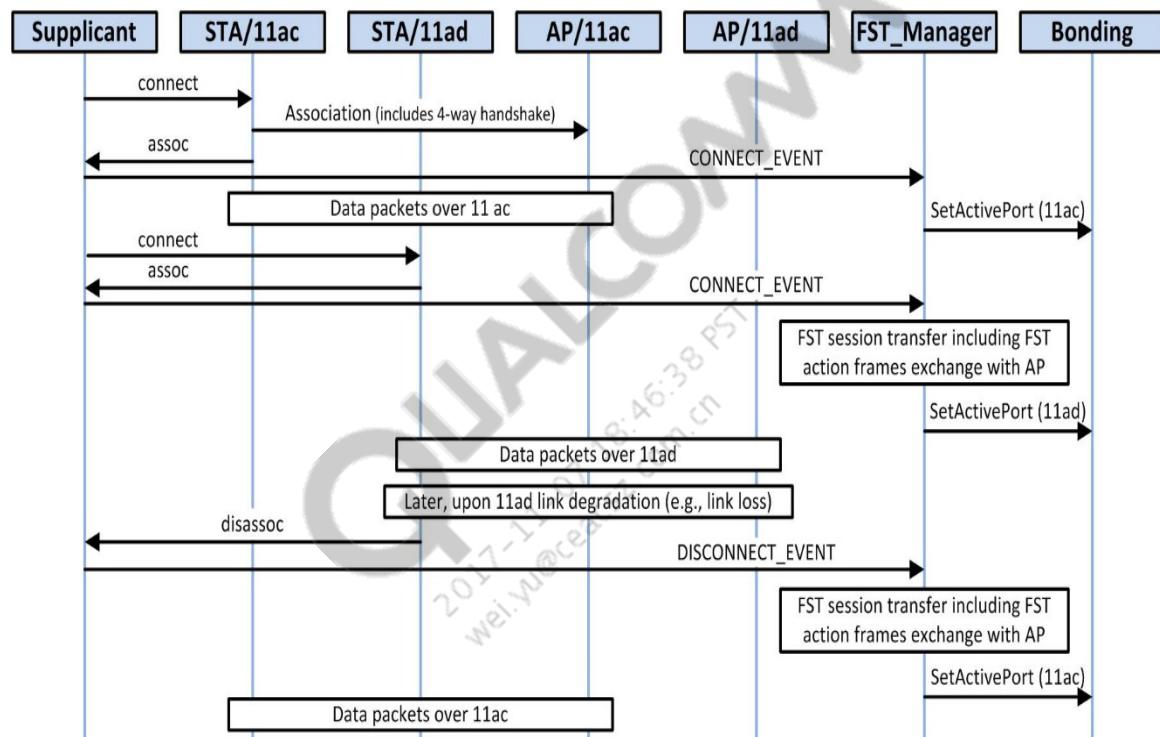
For example, assuming an FST client with 802.11ad active, a second with 802.11ad not active, and third non-11ad client, the FST manger configures the bonding driver such that:

- For the first client, transmit all packets (with destination MAC address matching the first client MAC address) through the 802.11ad interface
- For the second client, transmit all packets (with destination MAC address matching the second client MAC address) through 802.11n/802.11ac interface.
- For the third client, transmit all packets (with destination MAC address matching the third client MAC address) through 802.11n/802.11ac interface.

The data-switch policy is implemented by the FST manager. It configures the system to transfer data through the 802.11ad interface when an 802.11ad connection is available. If no 802.11ad connection is available, the FST manager configures the data to pass through the non-802.11ad interface. This configuration is done separately for each client.

## 21.6.2 FST flows

### 21.6.2.1 High level flow



**Figure 21-11 High-Level Flow when FST-Capable Clients Connect**

In this example shown, the client initially connects through 802.11ac. After the standard association flow, hostapd notifies the AP FST manager that a new client connected. FST manager configures the bond driver to route data packets with the client MAC address through the 802.11ac interface.

The same client initiates connection over 802.11ad, and FST manager identifies that both connections originate from the same FST-capable client (according to multiband IEs that are part of management frames). Because FST manager policy for session transfer is based on 802.11ad connect/disconnect events and 802.11ad just connected, FST manager instructs bonding driver to transfer packets to that client through the 802.11ad interface. Prior to configuring the bond driver, FST manager communicates with the client to notify the client about the desired switch through FST action frames as defined by the specification.

In the event that the 802.11ad connection drops, FST manager configures the bonding driver to transfer the data through its 802.11ac interface.

### 21.6.2.2 FST session flow

In the example shown in the following figure, fstman1/hostap1 run on the first peer while fstman2/hostap2 run on the second. Either peer can be either AP or STA because the flow is symmetrical. wlan0 and wlan1 refer to the Wi-Fi interfaces, and wlan1 has higher priority. The following figure describes these steps:

1. Initial association takes place on wlan0. On both sides, fstman is notified that the peer associated.
2. When association takes place on wlan1, fstman identifies that it is from same FST peer.
3. Because wlan1 has higher priority, fstman initiates FST session switch. The switch occurs through FST action frames sent between the peers, with the help of hostap.
  - a. Setup request is the first action frame sent and transmits over the current active band (wlan0).
  - b. Peer replies with setup response action frame over the current active band (wlan0).
  - c. ACK request and response action frames are exchanged between peers over new band (wlan1).
  - d. At this stage, fstman prepares a new FST session to switch from wlan1 to wlan0 when needed. The switch for this new session is not executed because wlan1 has higher priority than wlan0. The creation is done over the new band (wlan1) using FST setup request/response action frames.
4. When the peer disconnects over wlan1, the session that was already created takes effect through exchange of ACK request/response action frames over the wlan0 band.

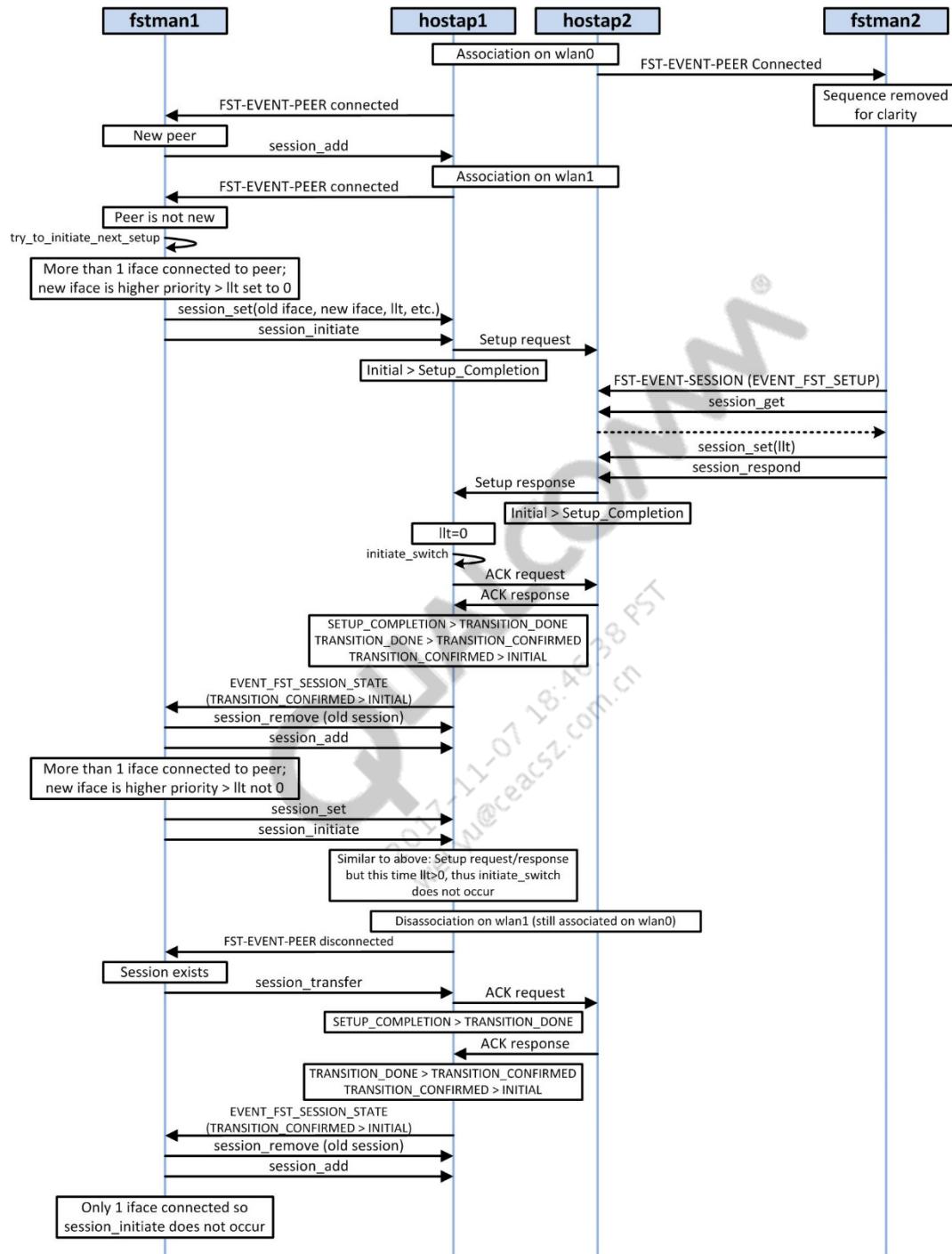
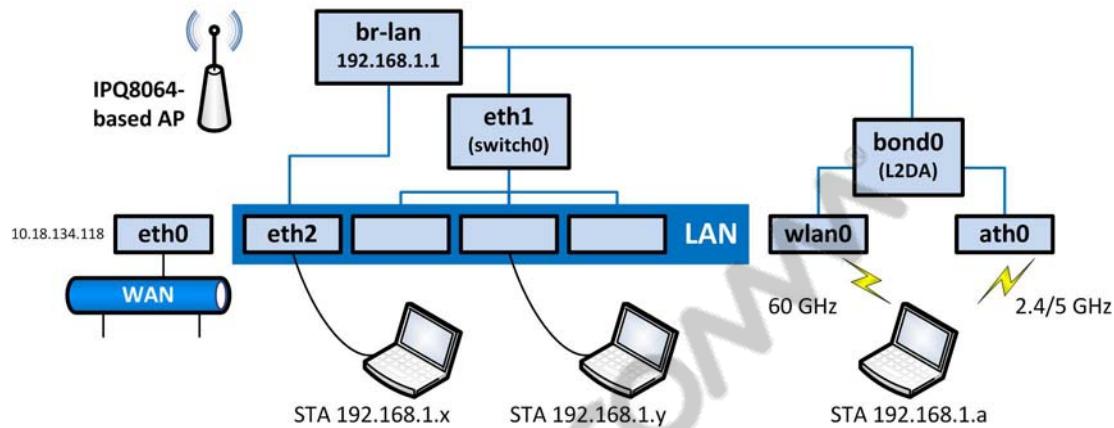


Figure 21-12 FST Session Flow (Control Path)

### 21.6.3 FST network topology



**Figure 21-13 FST network topology**

When FST is enabled, the interfaces participating in FST protocol are enslaved under the bond driver, and the bond driver is enslaved under br-lan. Enslaving under the bond driver should take place before hostapd starts running over the interfaces used for FST. The FST manager should start running after hostapd starts managing the interfaces used for FST.

### 21.6.4 FST manager

The **fst-manager** package is enabled by default for IPQ8064-based releases and FST manager code is included in the git project **wigig/wigig-utils**.

|                                   |                                                     |
|-----------------------------------|-----------------------------------------------------|
| Source code path                  | <code>qsdk/qca/src/wigig-utils/fst-manager</code>   |
| Package path (Enabled by default) | <code>qsdk/qca/feeds/wigig-utils/fst-manager</code> |

### 21.6.5 Debug logs

This section describes how to enable the detailed debug logs.

- For FST manager, edit the `/usr/sbin/fst.sh` script on target, modify this line:

```
/usr/sbin/fstman /var/run/hostapd/global &
```

To:

```
/usr/sbin/fstman -ddd /var/run/hostapd/global &
```

The log can be seen as part of logread. To see only FST specific logs, run:

```
logread -f | grep fstman > /var/fstman.log &
```

- For hostapd, edit the `/etc/init.d/qca-hostapd` script on target, modify this line:

```
hostapd -g /var/run/hostapd/global -B -P /var/run/hostapd-
global.pid
```

To:

```
hostapd -t -f /var/hostapd.log -g /var/run/hostapd/global -B -P
/var/run/hostapd-global.pid
```

This change enables verbose log of hostapd to a file that includes FST engine in hostapd. Note that hostapd must be compiled with CONFIG\_DEBUG\_FILE to have this enabled.

A common problem with FST is not having a successful FST switch. This usually happens due to failure of transmitting some FST action frames on either of the bands. The detailed logs show which actions frames fail.

### Example debug logs

This example shows a partial log output, which matches the flow shown in the *FST Session flow* section:

| Log Output                                                                                                                                                                  | Description                                                                                                                                              |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| hostapd : wlan1: AP-STA-CONNECTED 8c:fd:f0:02:02:be                                                                                                                         | hostapd received connection on wlan1                                                                                                                     |
| hostapd : FST-EVENT-PEER connected iface=wlan1 peer_ addr=8c:fd:f0:02:02:be                                                                                                 | hostapd notifies FST manager on new FST peer connected                                                                                                   |
| fstman : [00168859.285656] FST: CTRL: fst_ctrl_notify: recv: <3>FST-EVENT-PEER connected iface=wlan1 peer_ addr=8c:fd:f0:02:02:be                                           | FST manager received peer connected event                                                                                                                |
| fstman : [00168859.293115] FST: MGR: _fst_mgr_peer_try_ to_initiate_next_setup: peer 8c:fd:f0:02:02:be: session 0: initiating setup: old_iface=wlan0 new_ iface=wlan1 llt=0 | FST switches session by initiating new FST session (old interface wlan0, new interface wlan1)                                                            |
| hostapd : FST: 0 (0x00000000): [8c:fd:f0:02:02:be,8c:fd:f0:02:02:be] :wlan0: FST Action 'Setup Request' sent                                                                | hostapd sent <b>Setup Request</b> FST action frame over wlan0                                                                                            |
| hostapd : FST: bond0: wlan0: FST Action 'Setup Response' received!                                                                                                          | <b>Setup Response</b> FST action frame received on wlan0.                                                                                                |
| hostapd : FST: 0 (0x00000001): [8c:fd:f0:02:02:be,8c:fd:f0:02:02:be] :wlan1: FST Action 'Ack Request' sent                                                                  | <b>ACK Request</b> FST action frame sent on new interface (wlan1).                                                                                       |
| hostapd : FST: bond0: wlan1: FST Action 'Ack Response' received!                                                                                                            | <b>ACK Response</b> FST action frame received on wlan1.                                                                                                  |
| FST: MGR: _fst_mgr_on_ctrl_notification_state_change: session 0: state TRANSITION_DONE => TRANSITION_ CONFIRMED                                                             | FST manager received notification that transition was completed successfully. At this stage, FST manager instructs bond driver for the data-path switch. |

The logs include additional details information not shown in this example, although though are the main checkpoints to check within the log to determine the status of FST switch.

QUALCOMM®  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn

# 22 Logging and Statistics for Debugging

---

This chapter describes the utilities that can be used for diagnostic and troubleshooting purposes, such as packet logs, commands to display AP statistics, and WLAN event reports.

## 22.1 Packet Logging

### 22.1.1 Packet Logging for Direct Attach

Packet logging is a diagnostic feature which enables capturing information about the packets being received and transmitted by WLAN MAC. This information is useful for debugging throughput issues. The key information captured is the packet descriptor words that are exchanged between the software and hardware MAC. Additional information about packets includes timestamp, some parts of the header which includes the Frame Control and Sequence Control fields, and the last two bytes of the address fields. Apart from packet data, other information related to transmission such as rate control activity is also captured.

#### 22.1.1.1 Implementation

The events/data that are captured by the packet logging module and the API to log this data are listed below. These events are stored in a circular buffer whose size is configurable. Logging of events listed below can be enabled/disabled individually.

- Transmit Descriptors (Control and Status): Tx descriptors are logged from the transmit completion handlers in the LMAC by using the interfaces *ath\_log\_txctl* and *ath\_log\_txstatus*. Beacon descriptor are logging while queuing the frame since completion interrupts are not enabled for beacon frames.
- Receive Descriptors: Receive descriptors (status) are logged from the receive task using the interface *ath\_log\_rx*.
- TCP headers: The packet buffers are parsed while logging transmissions, and receive descriptors and TCP headers are logged if TCP packets are found.
- Rate Control information: When a packet is transmitted, the transmit rate information is provided by the rate-control module (*ath\_log\_rcfind*), and the transmit status information is given to the rate-control module after transmit completion (*ath\_log\_rcupdate*).
- Debug text: Any debug prints from the driver can be captured in the packet log using the interface *ath\_log\_text*.

lmac/ath\_pktlog/pktlog.c contains the core implementation of packet logging. Packet logging can be disabled during driver build by defining the compiler flag REMOVE\_PKT\_LOG. The

following APIs are provided by the packet logging module to implement the user CLI for configuring and using the packet logging tool:

- `pktlog_enable`—Enables/disables packet logging. The set of events to be logged should be specified. Packet logging will be disabled if no event is specified. The events to be enabled should be given as a bitmask. `lmac/ath_pktlog/pktlog_fmt.h` contains the definitions of these events.
- `pktlog_setsize`—Sets the size of the packet log buffer in bytes. Setting the size to zero will de-allocate the memory used to capture packet log data.
- `pktlog_read`—Copies the packet log data captured by driver into the user buffer.

The preceding APIs can be used by the OS shim layer to provide a user interface for using the packet logging tool. If multiple wireless radios are present on the AP, it is possible to capture log data for each adapter separately or as a single log file for the entire system. This data-capture is controlled by passing the softc handle of the radio to the aforementioned APIs. System-wide logging is performed if this handle is not provided. Apart from enabling/disabling the logging of events, certain useful options are available to stop the packet logging automatically, based on specific traffic criteria, which includes the following:

- Throughput below a threshold
- PER higher than a threshold
- Number of TCP SACK packets higher than a threshold

The OS shim layer should provide a user interface to configure these options and thresholds. `lmac/ath_dev/pktlog/linux.c` contains the Linux-specific implementation of an OS shim layer for packet logging.

### 22.1.1.2 Usage

The following is a sample usage of packetlog tool on a Linux AP. The packet log data can be retrieved through the procfs interface as shown here:

```
pktlogconf -e -s 2097152
```

This command enables packet logging and sets the size of log buffer as 2 MB. After waiting for the traffic to run, use the following commands to disable and copy the packetlog data to a file.

```
pktlogconf -d
cp /proc/ath_pktlog/system /tmp/pktlog.dat
```

The data captured by packet logging must be decoded using the Perl script, `owlidump.pl`. This script has various options to interpret the packetlog data. The list of options available can be found by running '`owlidump.pl -h`'. Running this script on the packetlog capture file without any options will list all the logged events/information in chronological order. Some useful reports can also be generated from the packetlog data, which are useful for analyzing throughput issues. For example, "`owlidump.pl -R`" generates a summary of Tx and Rx rates and the PER data as shown in the following log file.

```

owldump.pl -R pktlog.dat

File endianness: Big (AP)
File version: 10010
File size: 1048523 bytes
MAC Ver/Rev: 0x300/0x0 (Unknown)
Number of records: 13219
Duration of log: 0.169 seconds

-- Tx Rate Statistics --
Num Avg # Subframes Weighted Failures Total
Rate Frames Subfr % Good Bad PER PER RTS Data ERs PER
-----
M15h 233 22.3 100.00% 5192 0 0.00% 0.00% 0 17 0 7.17
-----
Total Data frames : 233
Total RTS failures : 0 (PER = 0.00%)
Total Data failures : 17 (PER = 6.80%)
Note: Excessive Retries (ERs) are accounted for by the last rate of the series

-- Rx Rate Statistics --
Num Avg # Subframes Weighted ACK/BA
Rate Frames Subfr % Good Bad PER PER Retries Fails <--- %
-----
M12 1 1.0 0.37% 1 0 0.00% 0.00% 0 0 0.00%
M13 6 22.8 2.21% 136 1 0.73% 0.02% 47 0 0.00%
M14 66 30.0 24.26% 1974 4 0.20% 0.05% 288 0 0.00%
M15 61 25.4 22.43% 1535 17 1.10% 0.25% 301 0 0.00%
M15h 138 26.2 50.74% 3533 77 2.13% 1.08% 478 0 0.00%
-----
Total Data frames : 272
Total A-MPDU Subframe PER : 1.36% (Good=7179, Bad=99)
Total retries : 1114 (15.52% of good frames)
ACK/BA (dup seq) failures : 0.00%

```

**Figure 22-1 Packet-Log PER/Rate Report**

### 22.1.2 Packet Logging for Offloaded Architecture

Pktlog tool captures the Tx and Rx descriptors in real time and provides rate statistics, RSSI statistics, aggregate information, data type description for the purpose of debugging PHY/ MAC layer issues.

The Pktlog tool essentially consists of three components: ath\_pktlog.ko (driver), pktlogconf (user space utility), and the pktlogdecoder\_11ac.pl perl script.

To perform Pktlog compilation: Check the build config script under build/scripts/<platform>/config.<platform> for the flag REMOVE\_PKT\_LOG, it should be set to zero to compile-in the pktlog.

Perform the following steps to successfully set up the pktlog for QCA9880:

### 22.1.2.1 Set up pktlog for x86 host

1. `sudo insmod /usr/local/lib/ath_pktlog.ko`
2. `sudo pktlogconf -etx -a wifi0` (for tx frames only capture), `sudo pktlogconf -erx -a wifi0` (for rx frames only capture), `sudo pktlogconf -e -a wifi0` (for all the frame types capture).
  - a. Alternatively, `sudo pktlogconf -e<tx,rx,rcu,rcf> -s <size in bytes> -a wifi0` can be used for setting the size.

**NOTE** Exercise caution regarding the memory limitations on the platforms executing the pktlog.

- b. To enable rate control, find and update information, use `pktlogconf -ercu,rcf -a wifi0`. Rcu and rcf can also be individually enabled using `pktlogconf -ercu -a wifi0` or `pktlogconf -ercf -a wifi0`.
3. Disable pktlog: `sudo pktlogconf -d wifi0`.
4. `sudo cp /proc/ath_pktlog/wifi0 <your_location>/pktlog.dat`
5. How to decode using the perl script

`perl pktlogdecoder_11ac.pl -a <your_location>/pktlog.dat` to decode the text section (all the information is logged per MSDU basis) of the Tx and Rx frames. The following are some features that are different from the previous pktlog perl script output:

- a. MSDUs within an MPDU are indicated with small ‘a’, last MSDU is always indicated as ‘A’ for non AMPDU and ‘A+’ for an AMPDU, MPDU frames in an aggregate are indicated as ‘A+’, and the last MPDU in an aggregate is indicated as ‘A’. The rest of the information in other columns will remain as found in the previous version of the pktlog.
- b. Rates column is identified as OFDM rates (only numbers), legacy rates (only numbers) and HT rates: identified as MXYY (where XX corresponds to the rate series and Y corresponds to the nss); VHT rates as vMXYY (where v is VHT, XX corresponds to the rate series and Y corresponds to the nss).
6. `perl pktlogdecoder_11ac.pl -R <your_location>/ pktlog.dat` to obtain the Rate statistics.
7. `perl pktlogdecoder_11ac.pl -S <your_location>/ pktlog.dat` to obtain the RSSI statistics.
8. `perl pktlogdecoder_11ac.pl -G <your_location>/ pktlog.dat` to obtain the Aggregate statistics.
9. `perl pktlogdecoder_11ac.pl -K <your_location>/ pktlog.dat` to obtain the PHY error statistics.
10. `perl pktlogdecoder_11ac.pl -V <your_location>/ pktlog.dat` to obtain the EVM statistics.

### 22.1.2.2 Set up pktlog for AP platform

The Steps 1 and 4 that are described in the procedure for setting up pktlog for x86 are different for AP platform:

1. apup will automatically bring up the ath\_pktlog.ko. If it does not bring up the pktlog, automatically, then perform the following steps:

- a. cd /lib/modules/2.6.31/net
- b. insmod ath\_pktlog.ko

2. Same as Step 2 in x86 without sudo

3. Same as Step 3 in x86 without sudo

4. cp /proc/ath\_pktlog/wifi0 /tmp/pktlog.dat

5. cd /tmp

**NOTE** Make sure that the AP platform is connected to the host over an Ethernet link and the TFTP server is running on the host.

6. Execute tftp -p <host\_ip> -l pktlog.dat on AP platform.

7. Decoding pktlog data using perl script on the host is similar to steps 5 to 10 as described under x86 pktlog setup.

## 22.2 AP Statistics

The application ‘apstats’ provides a number of statistics relevant to Access Point functionality. There are a large number of statistics which can be considered to fall under AP functionality. But the selection of statistics included under this application is driven by Enterprise AP requirements. The application can be expanded whenever new requests come in.

The statistics are arranged into four levels:

1. Overall AP level – WLAN and Ethernet
2. Per-WLAN radio level
3. Per-WLAN VAP level
4. Per-WLAN Node level

While apstats can be used directly by end users, it is also meant to serve as sample code to illustrate how various stats can be retrieved and used by customer written code (for example, SNMP implementation tailored to a customer-proprietary MIB definition).

Some of the stats are disabled by default at runtime. This is because they can have a slight performance impact. Further information is given in later sub-sections.

**NOTE** This application is different from athstats, though both applications share the use of some ioctls to retrieve data from the driver. The athstats utility contains a large amount of debugging data added over time by various developers (in addition to traditional stats). A lot of this data might not be relevant for Enterprise AP customers. Therefore,

apstats has been developed as a dedicated application to serve the specific needs of such customers.

### 22.2.1 AP Statistics Implementation

Most of the statistics gathered are maintained in the driver and incremented at appropriate points in the Tx/Rx path. The points of increment are not explicitly listed here since it is trivial to trace the required statistic. Additionally, the driver can optionally carry out Link Level Throughput, periodic (windowed) PER, and Channel Utilization measurements, if enabled at run time.

The code for Link Level Throughput and periodic (windowed) PER is present in *umac/base/ieee80211\_prdperstats.c*. It is included in the build only if the *UMAC\_SUPPORT\_PERIODIC\_PERFSTATS* compile time flag is enabled. For throughput measurement, the driver maintains a histogram of Tx and Rx byte counts (including 802.11 headers) over a window of the last n milliseconds (configurable). Similarly, for periodic PER measurement, it maintains another histogram for Tx successes and estimate of unsuccessful Tx tries (see note on PER in [Section 22.2.4](#)). These histograms are periodically updated. When the throughput or periodic PER value is requested, the appropriate histogram is consulted to calculate the value over the window.

The code for Channel Utilization measurement is present primarily in *umac/mlme/ieee80211\_bssload.c*. It is included in the build only if the *UMAC\_SUPPORT\_CHANUTIL\_MEASUREMENT* compile time flag is enabled. The function *ieee80211\_beacon\_chanutil\_update()* is used to update the channel utilization (CU) value and is shared with QBSS load related code which falls under the *UMAC\_SUPPORT\_BSSLOAD* compile time flag. *ieee80211\_beacon\_chanutil\_update()* is called as part of the procedures which update the dynamic parts of the beacon during beacon generation. This function updates the CU value every 10 beacon intervals. It obtains the current cycle count and Rx clear-low count from hardware. It computes the CU as ratio of (delta in Rx clear-low count from last update) to (delta in cycle count from last update), normalized to a range of 0-255 as specified in the 802.11 standard.

Details about the statistics (and some caveats) are provided in subsequent sections of this chapter.

The apstats tool builds a tree structure with the overall AP being the root of the tree. The AP entity has the wifiX instances and Ethernet as its children. Each wifiX has athX instances created under it as its children. Similarly, each athX has the STAs currently associated with it (at time of invocation of apstats) as its children.

The apstats tool works through the tree making ioctl calls to get most of the statistics from the driver. It also computes a few itself using information obtained from the ioctl calls for a given entity and its tree descendants (which is the main use of the tree structure). The WLAN statistics for the first level and overall AP level are all computed this way.

apstats mainly invokes the following ioctls to gather various statistics:

- *ATHR\_PHY\_CTRL\_IOC*: To get Ethernet Link Rate information.
- *ATHR\_GMAC\_CTRL\_IOC*: To get other Ethernet statistics.
- *SIOCGATHPHYSTATS*: To get WLAN PHY statistics.
- *SIOCGATHSTATS*: To get WLAN radio level statistics.
- *SIOCG80211STATS*: To get WLAN VAP level statistics.

- *IEEE80211\_IOCTL\_STA\_INFO*: To get rate information for associated STAs.
- *IEEE80211\_IOCTL\_STA\_STATS*: To get other statistics for associated STAs.

## 22.2.2 AP Statistics Usage

**NOTE** ‘X’ below stands for an instance of an interface. For example, wifiX can stand for wifi0, wifi1, and so on.

### 22.2.2.1 apstats command invocation

**apstats [Level] [[-i interface\_name] | [-m STA\_MAC\_Address]] [-R]**

One of four levels can be selected. The levels are as follows (from the top to the bottom of the hierarchy):

- Entire Access Point
- Radio
- VAP
- STA

The default is Entire Access Point. Information for sub-levels below the selected level can be displayed by choosing the -R (recursive) option (not applicable for the STA level, where information for a single STA is displayed).

#### Levels

##### **-a or --aplevel**

- Entire Access Point Level (WLAN and Ethernet)
- No Interface or STA MAC needs to be provided

##### **-r or --radiolevel**

- Radio Level
- Radio Interface Name needs to be provided

##### **-v or --vaplevel**

- VAP Level
- VAP Interface Name needs to be provided

##### **-s or --stalevel**

- STA Level
- STA MAC Address needs to be provided

#### Interface

If Radio Level is selected:

**-i wifiX or --ifname=wifiX**

If VAP Level is selected:

**-i athX or --ifname=athX**

**-i wlanX or --ifname=wlanX**

### STA Mac Address

Required if STA Level is selected:

**-m xx:xx:xx:xx:xx:xx or --stamacaddr xx:xx:xx:xx:xx:xx**

### Other Options

**-R or --recursive**

Recursive display

## 22.2.2.2 Configuration of optional statistics

Some AP statistics are runtime disabled by default. This is because gathering these statistics involves a small performance cost. Commands for configurations related to gathering of these statistics are described in the following subsections.

**NOTE** These statistics also need to be enabled at compilation time. See [Section 22.2.1](#) for details.

### In-driver throughput measurement

For in-driver throughput measurement, use the following commands:

**NOTE** The following commands are supported in WLAN driver, version 9.x, and are not available in WLAN driver, version 10.x and later.

**#iwpriv wifiX thrput\_enab val**

Activate/Deactivate in-driver throughput measurement.

If val is 1, then it is activated. If val is 0, then it is deactivated.

**#iwpriv wifiX get\_thrput\_enab**

Get status on whether in-driver throughput measurement is activated or not.

If 1 is returned, then it is activated. Alternatively, if 0 is returned, it is deactivated.

**#iwpriv wifiX get\_thrput\_win**

Get in-driver throughput measurement window (in milliseconds).

**#iwpriv wifiX thrput\_win val**

Set in-driver throughput measurement window (in milliseconds).

Min: 1000 Max: 30000 Default: 3000

Can be set only when the measurement is deactivated.

#### #iwpriv wifiX get\_thruput

(Not required if using apstats)

Get throughput value in kbps, measured in the driver in within the throughput measurement window. Note that only 802.11 data frames received/transmitted are considered for the measurement. **802.11 header bytes are included in the computation.**

If the measurement is deactivated, 0 is returned.

(If activated, 0 may also be returned if no data frames were transmitted/received in the last n milliseconds, where n = value of measurement window).

### PER measurement

For periodic PER measurement, use the following commands:

**NOTE** The following commands are supported in WLAN driver, version 9.x, and are not available in WLAN driver, version 10.x and later.

#### #iwpriv wifiX PER\_enab val

Activate/Deactivate periodic PER measurement.

If val is 1, then it is activated. If val is 0, then it is deactivated.

#### #iwpriv wifiX get\_PER\_enab

Get status on whether periodic PER measurement is activated or not.

If 1 is returned, then it is activated, else if 0 is returned, it is deactivated.

#### #iwpriv wifiX get\_PER\_win

Get periodic PER measurement window (in milliseconds).

#### #iwpriv wifiX PER\_win val

Set periodic PER measurement window (in milliseconds).

Min:3000 Max:900000 Default:300000

Can be set only when the measurement is deactivated.

#### #iwpriv wifiX get\_prdic\_PER

(Not required if using apstats)

Get periodic PER value (percentage).

If the measurement is deactivated, 0 is returned.

(If activated, 0 may also be returned if no frames were transmitted in the last n milliseconds, where n=value of measurement window).

#### **NOTE**

1. This value is computed considering number of unsuccessful frame transmission tries and total number of frame transmissions from this radio. Therefore, the value specifically indicates the quality (good or bad) of the channel for this radio to use in the configured measurement window. It does not give a generic indicator of channel quality (since other BSS transmissions are not considered).
2. If only a small number of frame transmission attempts occur in the window, and a large proportion of these attempts are retries, then the value returned is high.
3. This value is not used by the rate control algorithm (which maintains a different and detailed set of weighed average PER values per rate per STA).

For channel utilization measurement, use the following commands:

**NOTE** The following descriptions also apply to wlanX interfaces. The following commands are available in WLAN driver, version 9.x and later.

#### **#iwpriv athX chutil\_enab val**

Activate/Deactivate Channel Utilization measurement.

If val is 1, then it is activated. If val is 0, then it is deactivated.

#### **#iwpriv athX get\_chutil\_enab**

Get status on whether Channel Utilization measurement is activated or not.

If 1 is returned, then it is activated, else if 0 is returned, it is deactivated.

#### **#iwpriv athX get\_chutil**

(Not required if using apstats)

Get Channel Utilization value (0-255).

This is measured as defined in the 802.11 standard.

### **22.2.3 AP Statistics Listing**

The statistics currently provided by apstats are listed in the following subsections:

#### **22.2.3.1 Overall AP Level**

##### **WLAN**

1. No. of data frames transmitted.
2. No. of data bytes transmitted.
3. No. of data frames received.

4. No. of data bytes received.
5. No. of unicast data frames transmitted.
6. No. of multicast/broadcast data frames transmitted.
7. Average link rate of transmissions.
8. Average link rate at which frames were received.
9. Resource Utilization (0-255). This is the average of Channel Utilization figures across radios.
10. No. of PHY errors in frames received.
11. No. of CRC errors in frames received.
12. No. of MIC errors in frames received.
13. No. of decryption errors in frames received.
14. No. of receive errors.
15. No. of frames whose transmission failed. (Note: Not number of failed attempts).
16. Throughput in kbps. This is the sum of throughput values across radios.
17. PER measured from start of operation. This is the average of total PER values across radios.
18. Average of periodic PER values across all radios.

### Ethernet

1. Tx link rate.
2. Rx link rate.
3. No. of unicast frames received.
4. No. of unicast frames transmitted.
5. No. of multicast frames received.
6. No. of multicast frames transmitted.
7. No. of broadcast frames received.
8. No. of broadcast frames transmitted.
9. Total no. of frames received.
10. Total no. of bytes received.
11. Total no. of frames transmitted.
12. Total no. of bytes transmitted.
13. No. of Tx errors.

#### 22.2.3.2 Radio level

The following statistics are provided for each radio:

1. No. of data frames transmitted.
2. No. of data bytes transmitted.

3. No. of data frames received.
4. No. of data bytes received.
5. No. of unicast data frames transmitted.
6. No. of multicast/broadcast data frames transmitted.
7. No. of beacon frames transmitted.
8. No. of management frames transmitted (including beacon frames).
9. No. of management frames received.
10. Rx RSSI of last frame received.
11. Channel Utilization (0-255). This is the average of per-VAP values, considering those VAPs with the measurement enabled.
12. No. of PHY errors in frames received.
13. No. of CRC errors in frames received.
14. No. of MIC errors in frames received.
15. No. of decryption errors in frames received.
16. No. of receive errors.
17. No. of frames whose transmission failed. (Note: Not number of failed attempts).
18. Throughput in kbps
19. PER measured from start of operation.
20. Periodic PER measured in configured time window.

### 22.2.3.3 VAP level

The following statistics are provided for each VAP:

1. No. of data frames transmitted.
2. No. of data bytes transmitted.
3. No. of data frame payload bytes transmitted.
4. No. of data frames received.
5. No. of data bytes received.
6. No. of data frame payload bytes received.
7. No. of unicast data frames transmitted.
8. No. of multicast/broadcast data frames transmitted.
9. Average link rate of transmissions.
10. Average link rate at which frames were received.
11. No. of receive errors.
12. No. of frames whose transmission failed. (Note: Not number of failed attempts).

#### 22.2.3.4 Node level

The following statistics are provided for each STA currently associated with the AP at the time of invocation of apstats:

1. No. of data frames sent to STA.
2. No. of data bytes sent to STA.
3. No. of data frames received from STA.
4. No. of data bytes received from STA.
5. No. of unicast data frames sent to STA.
6. Average link rate used for transmissions to STA.
7. Average link rate used by STA to transmit to us.
8. No. of MIC errors in frames received from STA.
9. No. of decryption errors in frames received from STA.
10. No. of receive errors for this STA.
11. No. of frames destined to STA whose transmission failed (Note: Not number of failed attempts).
12. Rx RSSI of last frame received from this STA.

#### 22.2.4 Important Notes

1. Based on customer requirements, the WLAN statistics are mainly for 802.11 data frames.
  - a. The Tx/Rx packet and byte counts, Unicast and Multicast counts, Failed Tx packet counts (note that this is different from Tx retries), Tx/Rx payload counts, Rx MIC/Decryption errors, throughput measurement are all for 802.11 data frames.
  - b. This differentiation is not applied for Rx CRC and PHY errors, since it is difficult to classify such frames.
  - c. Total Rx errors is a composite count. The standard definition currently in use in QCA code across platforms is ( $rx\_tooshort + rx\_wepfail + rx\_decap\_err + rx\_nobuf + rx\_decryptcrc\_err + rx\_ccmpmic\_err + rx\_tkipmic\_err + rx\_tkipicv\_err$ )
  - d. Thus, the above differentiation regarding considering 802.11 data frames does not apply in this case.
  - e. The above differentiation does not apply for PER as well.
  - f. Applicability for other statistics should be self-evident.
2. WLAN side Tx/Rx data rate:  
Average values are given for link speeds used, not the most recent value. That is, this does not equal throughput. There is a separate stat for throughput.
3. Throughput measurement:  
The apstats application requires throughput measurement to be enabled for at least one of the radios, to provide an AP level throughput value. The AP level value is the sum of throughput values for all those radios for which the measurement is enabled.

PER calculation:

It would have been best to use the correct number of Tx retries and successes in the PER computation. However it is currently not possible to get accurate counts of Tx retries from the hardware. So after considering various alternatives, it has been decided to maintain a rough estimate of Tx unsuccessful tries in the driver (not including RTS), based on transmission series configured for each packet, and final transmission series index returned by hardware in the completion descriptor for each packet.

The PER is calculated as follows:

Rough estimate of Unsuccessful Transmissions/(Successful transmissions + Rough estimate of unsuccessful transmissions)

(The Qualcomm Technologies driver does maintain another set of PER values, but on a per-rate, per-STA basis using a variety of inputs and weighed ratio updates that are appropriate for per-rate calculation. It would be difficult to combine these individual PER values to represent a radio-wide PER value in a way that would make sense).

The AP level value is the average of per-radio values. Individual numerators and denominators are not used separately in determining the average. The per-radio ratios themselves are averaged. The reason is that failures and successes for each radio are not treated equally. Doing so can provide an inaccurate, pessimistic view of link quality present at the AP as a whole. Instead, each per-radio ratio is treated as a standalone indicator of the quality present at that radio, and these ratios are averaged for the AP.

The apstats application requires periodic PER measurement to be enabled for all radios, to provide an AP level periodic PER value.

a. Channel Utilization measurement:

Values are calculated and reported as given in the 802.11 standard, on a scale of 0-255.

Additionally, AP level resource utilization is the average of WLAN per-radio Channel utilization values.

The apstats application requires Channel Utilization measurement to be enabled for at least one VAP on every radio to provide an AP level Resource Utilization value. The AP level value is the average of per-radio values (which in turn are the average of per-VAP values considering those VAPs for which the measurement is enabled).

b. Stat reset capability is currently not provided in apstats as some of the stats may be used by some code sections in the driver and should not be reset. However, if it is required, it can be added for specific stats.

## 22.2.5 Sample apstats Output

The following is a sample output from apstats (taken in a very busy office environment):

```
~ # apstats -a -R
AP Level Stats:
WLAN Stats:
Tx Data Packets      = 4500
Tx Data Bytes        = 6602775
Rx Data Packets      = 2250
Rx Data Bytes        = 200833
```

|                                 |         |
|---------------------------------|---------|
| Tx Unicast Data Packets         | = 4316  |
| Tx Multi/Broadcast Data Packets | = 184   |
| Resource Utilization (0-255)    | = 135   |
| Average Tx Rate (kbps)          | = 69417 |
| Average Rx Rate (kbps)          | = 81781 |
| Rx PHY errors                   | = 193   |
| Rx CRC errors                   | = 2783  |
| Rx MIC errors                   | = 0     |
| Rx Decryption errors            | = 0     |
| Rx errors                       | = 0     |
| Tx failures                     | = 79    |
| Throughput (kbps)               | = 9073  |
| Total PER (%)                   | = 39    |
| PER over configured period (%)  | = 38    |

#### Ethernet Stats:

|                      |           |
|----------------------|-----------|
| Rx Link Rate (kbps)  | = 1000000 |
| Tx Link Rate (kbps)  | = 1000000 |
| Rx Unicast Packets   | = 4326    |
| Tx Unicast Packets   | = 2189    |
| Rx Multicast Packets | = 0       |
| Tx Multicast Packets | = 8       |
| Rx Broadcast Packets | = 0       |
| Tx Broadcast Packets | = 61      |
| Tx Errors            | = 0       |
| Rx Packets           | = 4326    |
| Rx Bytes             | = 6499922 |
| Tx Packets           | = 2258    |
| Tx Bytes             | = 167203  |

#### Radio Level Stats: wifi0

|                                 |           |
|---------------------------------|-----------|
| Tx Data Packets                 | = 2692    |
| Tx Data Bytes                   | = 3929100 |
| Rx Data Packets                 | = 1313    |
| Rx Data Bytes                   | = 120170  |
| Tx Unicast Data Packets         | = 2569    |
| Tx Multi/Broadcast Data Packets | = 123     |
| Channel Utilization (0-255)     | = 235     |
| Tx Beacon Frames                | = 1096    |
| Tx Mgmt Frames                  | = 1461    |
| Rx Mgmt Frames                  | = 8061    |
| Rx RSSI                         | = 43      |
| Rx PHY errors                   | = 190     |
| Rx CRC errors                   | = 2693    |
| Rx MIC errors                   | = 0       |
| Rx Decryption errors            | = 0       |
| Rx errors                       | = 0       |
| Tx failures                     | = 10      |
| Throughput (kbps)               | = 1799    |
| PER over configured period (%)  | = 53      |
| Total PER (%)                   | = 54      |

#### VAP Level Stats: ath0 (under radio wifi0)

|                       |         |
|-----------------------|---------|
| Tx Data Packets       | = 83    |
| Tx Data Bytes         | = 11607 |
| Tx Data Payload Bytes | = 9239  |

|                                 |          |
|---------------------------------|----------|
| Rx Data Packets                 | = 22     |
| Rx Data Bytes                   | = 2572   |
| Rx Data Payload Bytes           | = 1912   |
| Tx Unicast Data Packets         | = 22     |
| Tx Multi/Broadcast Data Packets | = 61     |
| Average Tx Rate (kbps)          | = 39268  |
| Average Rx Rate (kbps)          | = 125581 |
| Rx errors                       | = 0      |
| Tx failures                     | = 6      |

Node Level Stats: 00:03:7f:10:49:3b (under VAP ath0)

|                         |          |
|-------------------------|----------|
| Tx Data Packets         | = 22     |
| Tx Data Bytes           | = 2608   |
| Rx Data Packets         | = 22     |
| Rx Data Bytes           | = 2572   |
| Tx Unicast Data Packets | = 22     |
| Average Tx Rate (kbps)  | = 39268  |
| Average Rx Rate (kbps)  | = 125581 |
| Rx MIC Errors           | = 0      |
| Rx Decryption errors    | = 0      |
| Rx errors               | = 0      |
| Tx failures             | = 0      |
| Rx RSSI                 | = 45     |

VAP Level Stats: ath1 (under radio wifi0)

|                                 |           |
|---------------------------------|-----------|
| Tx Data Packets                 | = 2609    |
| Tx Data Bytes                   | = 3917493 |
| Tx Data Payload Bytes           | = 3839347 |
| Rx Data Packets                 | = 1291    |
| Rx Data Bytes                   | = 117598  |
| Rx Data Payload Bytes           | = 98040   |
| Tx Unicast Data Packets         | = 2547    |
| Tx Multi/Broadcast Data Packets | = 62      |
| Average Tx Rate (kbps)          | = 1921    |
| Average Rx Rate (kbps)          | = 57699   |
| Rx errors                       | = 0       |
| Tx failures                     | = 4       |

Node Level Stats: 00:03:7f:10:49:3e (under VAP ath1)

|                         |           |
|-------------------------|-----------|
| Tx Data Packets         | = 2547    |
| Tx Data Bytes           | = 3908430 |
| Rx Data Packets         | = 1291    |
| Rx Data Bytes           | = 117598  |
| Tx Unicast Data Packets | = 2554    |
| Average Tx Rate (kbps)  | = 1921    |
| Average Rx Rate (kbps)  | = 57699   |
| Rx MIC Errors           | = 0       |
| Rx Decryption errors    | = 0       |
| Rx errors               | = 0       |
| Tx failures             | = 0       |
| Rx RSSI                 | = 44      |

Radio Level Stats: wifi1

|                 |           |
|-----------------|-----------|
| Tx Data Packets | = 1808    |
| Tx Data Bytes   | = 2673675 |
| Rx Data Packets | = 937     |

```

Rx Data Bytes = 80663
Tx Unicast Data Packets = 1747
Tx Multi/Broadcast Data Packets = 61
Channel Utilization (0-255) = 35
Tx Beacon Frames = 1064
Tx Mgmt Frames = 1341
Rx Mgmt Frames = 6075
Rx RSSI = 9
Rx PHY errors = 3
Rx CRC errors = 90
Rx MIC errors = 0
Rx Decryption errors = 0
Rx errors = 0
Tx failures = 69
Throughput (kbps) = 7274
PER over configured period (%) = 24
Total PER (%) = 24
VAP Level Stats: ath2 (under radio wifi1)
Tx Data Packets = 1808
Tx Data Bytes = 2673675
Tx Data Payload Bytes = 2619557
Rx Data Packets = 937
Rx Data Bytes = 80663
Rx Data Payload Bytes = 74155
Tx Unicast Data Packets = 1747
Tx Multi/Broadcast Data Packets = 61
Average Tx Rate (kbps) = 118240
Average Rx Rate (kbps) = 71922
Rx errors = 0
Tx failures = 4

Node Level Stats: 00:03:7f:10:49:d8 (under VAP ath2)
Tx Data Packets = 1734
Tx Data Bytes = 2663128
Rx Data Packets = 867
Rx Data Bytes = 70216
Tx Unicast Data Packets = 1734
Average Tx Rate (kbps) = 101447
Average Rx Rate (kbps) = 129749
Rx MIC Errors = 0
Rx Decryption errors = 0
Rx errors = 0
Tx failures = 0
Rx RSSI = 59

Node Level Stats: 00:1e:65:55:13:04 (under VAP ath2)
Tx Data Packets = 13
Tx Data Bytes = 1548
Rx Data Packets = 70
Rx Data Bytes = 10447
Tx Unicast Data Packets = 13
Average Tx Rate (kbps) = 135033
Average Rx Rate (kbps) = 14095
Rx MIC Errors = 0
Rx Decryption errors = 0
Rx errors = 0

```

```

Tx failures = 0
Rx RSSI = 47

VAP Level Stats: ath3 (under radio wifi1)
Tx Data Packets = 0
Tx Data Bytes = 0
Tx Data Payload Bytes = 0
Rx Data Packets = 0
Rx Data Bytes = 0
Rx Data Payload Bytes = 0
Tx Unicast Data Packets = 0
Tx Multi/Broadcast Data Packets = 0
Average Tx Rate (kbps) = <NO STA>
Average Rx Rate (kbps) = <NO STA>
Rx errors = 0
Tx failures = 65

```

## 22.2.6 Athstats Enhancement for 802.11ac

### 22.2.6.1 Statistics List

This section lists the statistics for 11AC radio. These stats can be enabled by “athstats” cmd described later in this section. In order to enable these stats, athstats sends special command “**WMI\_REQUEST\_STATS\_CMDID**” to firmware (F/W). After that F/W periodically sends event “**WMI\_UPDATE\_STATS\_EVENTID**” to host driver. Host driver does housekeeping of these stats. Once stats is disabled by athstats cmd, current counter get reset.

| Statistics       | Comments                                  |
|------------------|-------------------------------------------|
| ast_be_xmit      | Number of Beacons Transmitted             |
| ast_be_nobuf     | No buffer for Beacon                      |
| ast_tx_buf_count | Number of Tx Buffer description           |
| ast_tx_packets   | Number of Tx packets                      |
| ast_rx_packets   | Number of Rx packets                      |
| ast_tx_mgmt      | Number of Tx management                   |
| ast_tx_nobuf     | Number of frames dropped due to no buffer |
| ast_tx_xretries  | Number of Hardware retries                |
| ast_rx_num_data  | Number of Rx data packets                 |
| ast_rx_num_mgmt  | Number of mgmt. packets received          |
| ast_rx_num_ctl   | Number of control rx packets              |
| ast_tx_rssi      | Tx RSSI of the last frame                 |
| ast_rx_rssi      | Rx RSSI of the last frame                 |
| ast_rx_bytes     | Number of R bytes                         |
| ast_tx_bytes     | Number of Tx bytes                        |
| tx_compaggr      | Number of aggregated Tx packets           |

|                 |                                                    |
|-----------------|----------------------------------------------------|
| rx_aggr         | Number Rx aggregated packets                       |
| tx_retries      | Number of sub frames retried                       |
| tx_xretries     | Number of packets dropped due to excessive retry.  |
| tx_bawadv       | Number of times the block ack window was advanced. |
| tx_compunaggr   | Number of Tx un-aggregated frame                   |
| ast_rxorn       | Number of Rx overruns                              |
| ast_txturn      | Number of Tx underrun                              |
| ast_txto        | Number of Tx device timeout                        |
| ast_mib         | Number of MIB interrupts                           |
| ast_rx_badcrypt | Number of received Frames with bad crypt           |
| ast_rx_badmic   | Number of received Frames with bad mic             |

### 22.2.6.2 Design and implementation

The statistics which are mentioned above in the list are all specific to radio level stats. Few of the stats earlier were part of athstats feature. Currently, Qualcomm Technologies is planning to support them as part of athstats.

The stats collection is split in 2 parts, one part of the stats is at HOST and other part of collection is done in the Firmware, this decision has evolved due to firmware not having much memory and therefore the stats collection is done much in the HOST. The following paragraphs describe the 2 parts:

1. The 1<sup>st</sup> part of statistics collection is to collect them in the Host driver. The firmware sends notification to the host on per PPDU basis. The RX and TX statistics therefore can be collected on per PPDU basis. For every PPDU received and sent, the stats are processed. All the collection of the stats is on the data path and there will be some performance penalty; therefore an iwpriv through which the stats collection can be enabled/disabled is provided.
2. The 2<sup>nd</sup> part of statistics collection is done through fetching the statistics from the firmware on interval basis. These statistics are feasible to maintain in the Firmware and on every interval the Firmware pushes these stats structure up to the HOST and the HOST receives this structure as a event. There is a handler in the host which is registered to this event and this handler updates the HOST stats structure.
3. The user programmer queries the host driver for the stats through a well defined system call. When the system call is made the driver copies the stats structure to the user specified location.

### 22.2.6.3 Required CLIs to test the feature

As almost all of the statistics collection is done on the data path, which can be resource intensive. Therefore, an iwpriv to enable and disable the statistics is provided. By default it is disabled. An iwpriv must be issued to enable the statistics collection. The iwpriv to enable the statistics is “*iwpriv ath0 enable\_ol\_stats 1/0(enable/disable)*”.

The command to see the statistics is “*athstats -i wifiI*”, where the “-i” flag provides the interface and also signifies whether the statistics of the direct-attach or offload path must be fetched.

## 22.3 Tracking latency and loss of 802.11 packets

To monitor and track successful delivery of every packet to the client through the AP, the following statistical parameters are useful:

- Completion of every packet –Timestamp of when the ACK was received, sequence number if it's a block ACK.
- Latency – A starting timestamp in the sk\_buff to be used as a reference point to calculate a time difference.
- Lost packets information – Loss rate count.

To enable tracking and loss of 802.11 packets, the following new fields are included per-ppdu statistics.

- PPDU ACK TIME STAMP – Timestamp of the PPDU acknowledgment
- PPDU Block Ack bitmap – PPDU bitmap enqueued, tried, and failed information

With an AP and STA set up, and an infra-client connected, ping tests and iperf are run between the AP and STA. OL statistics are enabled and Tx completion handler statistics are retrieved by running the `iwpriv wifiN enable_ol_stats 1` and `iwpriv wifiN data_txstats 1` commands respectively. The following is an example of the per-ppdu statistics with the PPDU acknowledgment timestamp and PPDU block acknowledgment bitmap fields included:

```
[12424.031755] ==radio=wifi1, STA=48:51:b7:19:97:bd
[12424.035412] ==ts->ppdu_rate=0x85 preamble=2, nss=0, mcs=5
[12424.040869] ==ts->ppdu_num_mpduSuccess=0x1
[12424.045118] ==ts->ppdu_num_mpduFail=0x0
[12424.049116] ==ts->ppdu_num_msduSuccess=0x1
[12424.053368] ==ts->ppdu_bytes_success=0x6e
[12424.057363] ==ts->ppdu_duration=0x7c
[12424.060925] ==ts->ppdu_retries=0x0
[12424.064300] ==ts->ppdu_is_aggregate=0x1
[12424.068110] ==ts->ts_flags=0x70
[12424.071234] ==ts->msdu_length=86
[12424.074452] ==ts->msdu_priority=0
[12424.077763] ==ts->ch_width=0
[12424.080606] ==ts->msdu_q_time=130 ms
[12424.084202] ==ts->ppdu_ack_timestamp=2742504465 usec
[12424.089135] ==ts->starting seq_number=280
[12424.093135] Bitmap of packets enqueued to HW, sent OTA and packets failed (343...280)
[12424.100756] ==ts->ppdu_bitmap_enqueued
[12424.104505] 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0001
==ts->ppdu_bitmap_tried
[12424.122061] 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0001
==ts->ppdu_bitmap_failed
[12424.139648] 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000
```

## 22.4 Enhanced station statistics display for Tx and Rx frames

Use the apstats -s -m xx:xx:xx:xx:xx:xx (MAC address of station) command to get all STA statistical details, including the total Tx unicast frame counter and length in bytes.

The following metrics are displayed in the output of this command:

- Total Rx uni-cast frame count and length in bytes
- Total Rx multi-cast frame count and length in bytes
- Total Tx uni-cast frame count and length in bytes
- Total Tx multi-cast frame count and length in bytes

The following is the sample CLI output:

```
root@OpenWrt:/# apstats -s -m a0:18:28:9f:09:38
Tx Data Packets(valid for offload driver) = 885
Tx Data Bytes(valid for offload driver) = 100234
Rx Data Packets = 4126
Rx Data Bytes = 186356
Tx Unicast Data Packets = 885
Average Tx Rate (kbps) = 325000
Average Rx Rate (kbps) = 292000
Last tx rate = 325000
Last rx rate = 292000
Last mgmt rx rate = 6000
Rx MIC Errors = 0
Rx Decryption errors = 0
Rx errors = 0
Packets Queued = 906
Host Discard = 0
Tx Dropped(valid for offload driver) = 21
Rx RSSI = 20
Rx MGMT RSSI = 20
```

The following is the sample output of the 80211stats command:

```
root@OpenWrt:/# 80211stats -a -i ath0
34:2:86:a:a1:f4:
    rx_data 518 rx_mgmt 12 rx_unicast 299 rx_multicast 219 rx_bytes 61856 rx_
    unicast_bytes 31258 rx_multicast_bytes 30598 rx_null 4 rx_null_bytes 112 rx_
    mgmt_bytes 454
        tx_data 248 tx_data_success 248 tx_unicast 248 tx_bytes 19543 tx_bytes_
    success 27975 tx_unicast_bytes 27975
        tx_assoc 2
```

By default “wifi\_nss\_olcfg” is configured with 3 (radios 0 and 1 are enabled for NSS Wi-Fi offload; therefore NSS\_offload is enabled in current code branch. Disable NSS offload using the “uci set wireless.qcawifi=qcawifi; uci set wireless.qcawifi.nss\_wifi\_olcfg=0”, UCI commands or changing “/lib/wifi/wifi\_nss\_olcfg” with 0.

```
root@OpenWrt:/# cat /lib/wifi/wifi_nss_olcfg 3
```

The code flow to initialize NSS and register callbacks in the “qca-wifi-10.4/os/linux/src/osif\_nss\_wifio\_if.c” source file is as follows:

```
int osif_nss_ol_wifi_init(void *hifctx) {
    ...
}
```

```

nss_wifiol_ctx = nss_register_wifi_if(ifnum,
(nss_wifi_callback_t)osif_nss.ol.wifi_ce_callback_func,
(nss_wifi_callback_t)osif_nss.ol.wifi_ext_callback_func,
(nss_wifi_msg_callback_t)osif_nss.ol.event_receive,
(struct net_device *)sc, features
...
}

```

Only after the `iwpriv wifi0 enable_ol_stats 1` command is issued to enable station statistics, the `apstats -s -m` contains outputs. The output is executed by “`ol_ath_enable_ap_stats()`”.

Per codes of “`ol_ath_enable_ap_stats()`”, RX statistic is enabled with one WDI event and the specific callback is “`process_tx_stats()`”, TX statistic is enabled with one DMI event, or WDI event similar to the RX, or one package log event.

```

int ol_ath_enable_ap_stats(struct ieee80211com *ic, u_int8_t stats_cmd) {
...
    STATS_RX_SUBSCRIBER.callback = ol_ath_get_all_stats;
    if(wdi_event_sub(scn->pdev_txrx_handle,
                     &STATS_RX_SUBSCRIBER,
                     WDI_EVENT_RX_DESC_REMOTE)) {
        return A_ERROR;
    }
    if (scn->is_ar900b) {
        if(wmi_unified_pdev_set_param(scn->wmi_handle,
                                       WMI_PDEV_PARAM_EN_STATS, 1) != EOK ) {
            return A_ERROR;
        }
    } else {
        STATS_TX_SUBSCRIBER.callback = ol_ath_get_all_stats;
        if(wdi_event_sub(scn->pdev_txrx_handle,
                         &STATS_TX_SUBSCRIBER,
                         WDI_EVENT_OFFLOAD_ALL)) {
            return A_ERROR;
        }
    }
#if defined(CONFIG_AR900B_SUPPORT) || defined(CONFIG_AR9888_SUPPORT)
    types |= WMI_PKTLOG_EVENT_TX;
    len = sizeof(wmi_pdev_pktlog_enable_cmd);
    buf = wmi_buf_alloc(scn->wmi_handle, len);
    ...
    cmd = (wmi_pdev_pktlog_enable_cmd *)wmi_buf_data(buf);
    cmd->evlist = WMI_PKTLOG_EVENT_TX;
    /*enabling the pktlog for stats*/
    if(wmi_unified_cmd_send(scn->wmi_handle, buf, len,
                           WMI_PDEV_PKTLOG_ENABLE_CMDID)) {
        return A_ERROR;
    }
#endif
    scn->scn_stats.ap_stats_tx_cal_enable = 1;
    ol_txrx_enable_enhanced_stats(scn->pdev_txrx_handle);

#endif QCA_NSS_WIFI_OFFLOAD_SUPPORT

```

```

        osif_nss.ol_stats_cfg(scn->hif_sc, stats_cmd);
#endif
...

```

Then the corresponding handler to the RX/TX WDI event is called in “ol\_ath\_get\_all\_stats()”.

```

void ol_ath_get_all_stats(...) {
    switch(event) {
        case WDI_EVENT_RX_DESC_REMOTE:
            if(process_rx_stats(pdev, log_data, peer_id, status)) {
                adf_os_print("Unable to process RX info\n");
                return;
            }
            break;

        case WDI_EVENT_TX_STATUS:
        case WDI_EVENT_OFFLOAD_ALL:
            if(process_tx_stats(pdev, log_data, peer_id, status)) {
                adf_os_print("\n Unable to process TX info \n");
                return;
            }
            break;

        default:
            break;
    }
}

```

### **Macro UMAC\_SUPPORT\_APONLY**

By default, it's enabled clearly in “os/linux/configs/config.wlan.unified.profile” and then forwarded to “os/linux/BuildCaps.inc”.

For generic case, it should be changed into disabled clearly in “os/linux/configs/config.wlan.unified.profile”.

### **Macro UMAC\_SUPPORT\_TXDATAPATH\_NODESTATS**

It functions for both AP-only case and generic case, by default it's disabled clearly “os/linux/configs/config.wlan.unified.profile” and then is forwarded to in “os/linux/BuildCaps.inc”.

To get statistic counters for “ns\_tx\_data\_success” and “ns\_tx\_bytes\_success”, it should be changed into enabled clearly in “os/linux/configs/config.wlan.unified.profile”.

## **22.5 Customer WLAN event reports**

This section describes the support that is introduced for several customer WLAN events. The following table describes all WLAN events that are supported:

| Event Name                | Description                                   |
|---------------------------|-----------------------------------------------|
| IEEE80211_EV_AUTH_IND_AP  | Event when AP received AUTH request from STA  |
| IEEE80211_EV_ASSOC_IND_AP | Event when AP received ASSOC request from STA |

|                                     |                                                                                   |
|-------------------------------------|-----------------------------------------------------------------------------------|
| IEEE80211_EV_REASSOC_IND_AP         | Event when AP received REASSOC request from STA                                   |
| IEEE80211_EV_DEAUTH_IND_AP          | Event when AP received DEAUTH request from STA                                    |
| IEEE80211_EV_DISASSOC_IND_AP        | Event when AP received DISASSOC request from STA                                  |
| IEEE80211_EV_KETSET_DONE_IND_AP     | Event when AP set WPA/WPA2 key finished                                           |
| IEEE80211_EV_MIC_ERR_IND_AP         | Event when AP received a packet with Error MIC                                    |
| IEEE80211_EV_BLKLST_STA_AUTH_IND_AP | Event when AP received AUTH request from a STA, and this STA is in AP's blacklist |

WLAN driver has previously implemented several events, and the basic event structure exists. To meet customer needs, the existing events are modified and several new events are added to WLAN driver. The following is an example of an event report:

```

union iwreq_data wrqu;
    struct ev_contact contact_buf;

    IEEE80211_ADDR_COPY(contact_buf.sta_addr, macaddr); =>Fill STA's mac
address
    IEEE80211_ADDR_COPY(contact_buf.ap_addr, vap->iv_myaddr);=>Fill AP's
mac address
    memset(&wrqu, 0, sizeof(wrqu));
    wrqu.data.flags = IEEE80211_EV_REASSOC_IND_AP; =>Event flag
    wrqu.data.length = sizeof(struct ev_contact);
    WIRELESS_SEND_EVENT(dev, IWEVCUSTOM, &wrqu, (char *)&contact_
buf); =>Report to Application

    struct ev_contact {
        u_int8_t sta_addr[IEEE80211_ADDR_LEN]; /* mac addr */
        u_int8_t ap_addr[IEEE80211_ADDR_LEN]; /* mac addr */
        u_int32_t arg;
    };
}

```

struct ev\_contact{} is used for carry driver event information to application layer. Here we define three fields. sta\_addr[] is 6 bytes, use for restore STA's mac address. ap\_addr[] also is 6 bytes length, use for restore AP's mac address. arg is for restore some special parameter, IEEE80211\_EV\_DEAUTH\_IND\_AP and IEEE80211\_EV\_DISASSOC\_IND\_AP use this field restore the deauth/disassoc reason flag. IEEE80211\_EV\_MIC\_ERR\_IND\_AP use this field restore keyid flag.

All event reports are implemented in osif\_umac.c. The customer\_event\_test.c file is used for testing customer event.

Copy it to os/linux/tools/, and add this file to Makefile. After compilation, an executable file **customer\_event\_test** is created. Copy the **customer\_event\_test** (using TFTP) to your testing board and start testing. This command does not require any input parameter, and instead, it remains in listen status. The following is the usage sample:

```

root@OpenWrt:/tmp# tftp -g 192.168.1.200 -r customer_event_test
root@OpenWrt:/tmp# chmod +x customer_event_test
root@OpenWrt:/tmp# ./customer_event_test &
root@OpenWrt:/tmp# start xiaomi event listen.....

```

The following are the different types of WLAN driver events:

- IEEE80211\_EV\_AUTH\_IND\_AP and IEEE80211\_EV\_ASSOC\_IND\_AP—This event is triggered when STA attempts to connect to the test AP.

```
root@OpenWrt:/tmp# ./customer_event_test &
root@OpenWrt:/tmp# start xiaomi event listen.....
root@OpenWrt:/tmp# Received IEEE80211_EV_AUTH_IND_AP event from STA(38:ca:da:0d:c0:99) to AP(8c:fd:f0:02:21:23)
Received IEEE80211_EV_ASSOC_IND_AP event from STA(38:ca:da:0d:c0:99) to AP(8c:fd:f0:02:21:23)
```

- IEEE80211\_EV\_DISASSOC\_IND\_AP—This event is triggered when STA connected to test AP and then connects to another AP.

```
root@OpenWrt:/tmp#
root@OpenWrt:/tmp# Received IEEE80211_EV_DISASSOC_IND_AP event from STA(38:ca:d:0d:c0:99) to AP(8c:fd:f0:02:21:23),the reason is 8
```

- IEEE80211\_EV\_DEAUTH\_IND\_AP—This event is triggered when STA connects to a to test AP and then disconnects from this AP or connects to another AP.

```
root@OpenWrt:/tmp#
root@OpenWrt:/tmp# Received IEEE80211_EV_DEAUTH_IND_AP event from STA(3c:a9:f4:b6:21:c8) to AP(8c:fd:f0:02:21:23),the reason is 1
```

- IEEE80211\_EV\_REASSOC\_IND\_AP—This event is triggered when reference AP and test AP are configured with same Wi-Fi (same SSID, same channel, and same encryption). Consider a scenario to bring-up reference AP, and allow STA to connect to reference AP. Bring up test AP, and bring down reference AP. STA reassociates with test AP.

```
root@OpenWrt:/tmp# Received IEEE80211_EV_AUTH_IND_AP event from STA(38:ca:da:0d:c0:99) to AP(8c:fd:f0:02:21:23)
Received IEEE80211_EV_REASSOC_IND_AP event from STA(38:ca:da:0d:c0:99) to AP(8c:fd:f0:02:21:23)
```

- IEEE80211\_EV\_BLKLST\_STA\_AUTH\_IND\_AP—This event is triggered when STA MAC is added to AP's blacklist, and then STA connects to test AP.

```

root@OpenWrt:/tmp#
root@OpenWrt:/tmp# iwpriv ath0 get_maccmd
ath0      get_maccmd:0
root@OpenWrt:/tmp# iwpriv ath0 maccmd 2
root@OpenWrt:/tmp# iwpriv ath0 addmac 3c:a9:f4:b6:21:c8
root@OpenWrt:/tmp# iwpriv ath0 getmac
ath0      getmac:3C:A9:F4:B6:21:C8
root@OpenWrt:/tmp# Received IEEE80211_EV_BLKLST_STA_AUTH_IND_AP event, STA(3c:a9:f4:b6:21:c8) is in blacklist of AP(8c:fd:f0:02:21:23)
Received IEEE80211_EV_AUTH_IND_AP event from STA(3c:a9:f4:b6:21:c8) to AP(8c:fd:f0:02:21:23)

```

- IEEE80211\_EV\_KEYSET\_DONE\_IND\_AP—This event is triggered with normal WPA or with WPA that is in connecting state. Bring up test AP with WPA/WPA2 mode. Allow the STA to connect to test AP with correct encryption and password.

```

Received IEEE80211_EV_AUTH_IND_AP event from STA(38:ca:da:0d:c0:99) to AP(8c:fd:f0:02:21:23)
Received IEEE80211_EV_ASSOC_IND_AP event from STA(38:ca:da:0d:c0:99) to AP(8c:fd:f0:02:21:23)
Received IEEE80211_EV_KEYSET_DONE_IND_AP event, AP(8c:fd:f0:02:21:23) finished key set to STA(38:ca:da:0d:c0:99)

```

IEEE80211\_EV\_MIC\_ERR\_IND\_AP—This event is triggered when test AP is received one packet with MIC error. Because the MIC error packet cannot be created, this event cannot be verified.

## 22.6 AP Diagnostics for Carrier

The primary focus of this feature is to monitor the Wi-Fi PHY transmit data rate of each connected clients. If the data rate falls below a configured threshold, a log is generated capturing event time and parameter values for the following list of parameters:

1. Type of event (Warning, Error, Normal)
2. Configured Wi-Fi PHY transmit data rate threshold (valid only for warning, error types)
3. MAC address of client
4. Power level in dB of signal to client
5. Client SNR
6. Transmit 802.11 version(802.11n/802.11ac)
7. Transmit channel index
8. Current Wi-Fi PHY transmit data rate
9. Transmit MCS index
10. Transmit channel width

11. Transmit guard interval
12. Transmit spatial stream number

#### Event type details

1. Warning - Data rate is lower than configured warning data rate threshold.
2. Error – Data rate is lower than configured error data rate threshold.
3. Normal – Data rate is higher than both error and warning thresholds.

Diagnostics feature makes use of acfg netlink backbone to send events to upper layer, acfg support for driver must be enabled to successfully send diagnostics event to upper layer. For this reason compiler directives ACFG\_NETLINK\_TX and UMAC\_SUPPORT\_ACFG should be defined for diagnostics feature to work normally.

### 22.6.1 Configuration

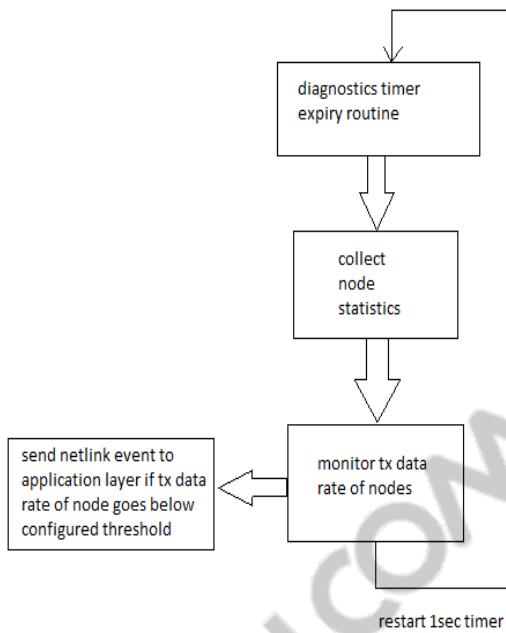
The following commands are available for diagnostics configuration:

1. iwpriv wifiN set\_diag\_enable <0/1>—Enables diagnostics (1) or disables diagnostics (0)
2. iwpriv wifiN get\_diag\_enable—Returns current status of diagnostics
3. iwpriv athN set\_err\_thres <data rate in mbps>—Sets error condition data rate threshold
4. iwpriv athN get\_err\_thres—Returns recently configured error condition data rate threshold
5. iwpriv athN set\_warn\_thres <data rate in mbps>—Sets warning condition data rate threshold
6. iwpriv athN get\_warn\_thres—Returns recently configured warning condition data rate threshold

### 22.6.2 High-level design

Diagnostics implementation for both architectures—direct attach and offload—can be broadly divided into the following two segments:

1. Diagnostics monitor client statistics
2. Diagnostics notify



### 22.6.2.1 Diagnostics monitor client statistics

A timer, acfg\_diag\_timer, configured to run every 1000 milliseconds iterates through the node table. This timer collects the above mentioned stats for each node and form a list (statistics list). This timer is not configurable.

A routine iterates through statistics list and collects all the entries whose data rate is lower than the configured warning or error threshold. These entries are then added into a new list (monitor list) for monitoring purpose. If an entry already exists in the monitor list then its statistics parameters are updated and it is marked to be notified to upper layer based on certain conditions. All the new entries are marked to be notified to upper layer.

The following are the conditions that cause an entry in monitor list to be notified to upper layer:

- A new entry.
- The data rate of node corresponding to the entry is lower than warning/error threshold for more than 15 seconds.
- The data rate of node corresponding to the entry is lower than warning/error threshold for more than 1 hour.
- The data rate of node corresponding to the entry has moved from below warning threshold to below error threshold and vice-versa.
- The data rate of node corresponding to the entry is greater than both the warning and error thresholds.

All other entries whose data rates are higher than the threshold are discarded. However, before discarding, these entries are checked to see if any of them belong to monitor list. If so, this match

indicates that the data rate of node corresponding to this entry is above both the thresholds and it is marked to be notified to upper layer.

**NOTE** The method of collection of statistics for nodes differs between direct attach and offload architectures. For direct attach, the required statistics from Rx data path and Tx completion path are collected. For offload, the statistics from peer WMI event handler are collected.

**NOTE** The firmware might not send the statistics of all the connected nodes in one event due to memory limitations (maximum of two nodes per event). Several events might be required to retrieve the statistics of all connected nodes if more than two nodes are connected at a time. This method of statistics-retrieval might cause a delay in updating of node statistics by the acfg\_diag\_timer. Because of this delay, the notification to upper layer about change in data rate might also be delayed.

### 22.6.2.2 Diagnostics notify

A routine iterates through monitor list and picks all the entries whose notify flag is set. For each such entries it sends an acfg netlink event to upper layer with all the collected statistics details. This event is captured by acfg\_tool application and logged to file called acfg\_event\_log.

## 22.7 Generate target core dump

This section describes the functionality to generate runtime target core dump. At runtime of an AP, users can generate a dump of the target core into a file without impacting the working of the AP. The core dump is used for troubleshooting errors and diagnosis purposes. The core dump at runtime is exactly the same as when a target assert happens.

An iwpriv command is introduced to dump the target at runtime. Enter the `iwpriv wifi0 dump_target 1` to dump target core.

This command calls the same function that is called when target crash happens so that the same core dump file can be recorded. The following is an example of code:

```
void ol_ath_dump_target(struct ol_ath_softc_net80211 *scn, int op)
{
    char fw_dump_file[128]={0};

    get_next_dump_file_index(scn, fw_dump_file);
    printk("## STARTING DUMP target to %s, op=%d\n", fw_dump_file, op);
    fw_get_core_dump(scn, fw_dump_file, op);
    printk("### TARGET ASSERT DUMP COLLECTION COMPLETE ***\n");
}

case OL_ATH_PARAM_DUMP_TARGET:
    ol_ath_dump_target(scn, *(int *)buff);
    break;
```

## 22.8 Enhanced client statistics display for RSSI, Tx/Rx packets, and STBC

This section describes the display of enhanced statistics, such as the RSSI of devices, Tx and Rx bytes and packets per second, and the space time block coding (STBC) capabilities, of connected clients. Use the apstats -s -m xx:xx:xx:xx:xx:xx (MAC address of station) command to view these enhanced client or STA statistical details. The following additional statistics are displayed:

| Type                           | Parameter                        | Description                                             |
|--------------------------------|----------------------------------|---------------------------------------------------------|
| RSSI                           | WifiMgmtClientStatsRSSI          | RSSI of client device                                   |
| TX/RX Bytes/Packets per second | WifiMgmtClientStatsRxBytesPerSec | Received bytes per second of the connected client.      |
|                                | WifiMgmtClientStatsRxPktsPerSec  | Received packets per second of the connected client.    |
|                                | WifiMgmtClientStatsTxBytesPerSec | Transmitted bytes per second of the connected client.   |
|                                | WifiMgmtClientStatsTxPktsPerSec  | Transmitted packets per second of the connected client. |
| Capability                     | WifiMgmtClientStatsSTBC          | STBC capabilities of the connected client.              |
|                                | WifiMgmtClientStatsMIMO          | Spatial streams of the connected client.                |
|                                | WifiMgmtClientStatsBandwidth     | Bandwidth of the connected client.                      |

### 22.8.1 Data structure changes

The following related variables are added:

```
struct ieee80211_nodestats {
    ...
    #ifdef ATH_SUPPORT_EXT_STAT
    + u_int32_t ns_tx_bytes_rate;           /* transmitted bytes per second */
    + u_int32_t ns_tx_data_rate;            /* transmitted data per second */
    + u_int32_t ns_rx_bytes_rate;           /* received bytes per second */
    + u_int32_t ns_rx_data_rate;            /* received data per second */
    #endif
    ...
}

struct ieee80211req_sta_info {
    ...
    #ifdef ATH_SUPPORT_EXT_STAT
    + u_int8_t isi_tx_stbc;                /* tx stbc */
    + u_int8_t isi_rx_stbc;                /* rx stbc */
    + u_int8_t isi_chwidth;                /* communication band width */
    #endif
    ...
}

typedef struct _nodelevel_stats_t
```

```

{
    ...
+ifdef ATH_SUPPORT_EXT_STAT
+    u_int64_t tx_bytes_rate;           /* transmitted bytes per second */
+    u_int64_t tx_data_rate;          /* transmitted data per second */
+    u_int64_t rx_bytes_rate;          /* received bytes per second */
+    u_int64_t rx_data_rate;          /* received data per second */
#endif

...
u_int8_t rx_rssi;                      /**< Rx RSSI of last frame received from
   this STA. */

u_int8_t rx_mgmt_rssi;                  /**< Rx RSSI of last mgmt frame received
   from
   this STA. */

+ifdef ATH_SUPPORT_EXT_STAT
+    u_int8_t tx_stbc;               /* tx stbc of this STA */
+    u_int8_t rx_stbc;               /* rx stbc of this STA */
+    u_int8_t chwidth;              /* communication band width with this
   STA */
#endif

...
}

typedef struct ieee80211com {
    ...
+ifdef ATH_SUPPORT_EXT_STAT
+    os_timer_t calculate the realtime rate*/      ic_client_stat_timer; /* timer to
   ...
+endif
}

```

To obtain the RSSI value, the following code is added:

```

static void
get_sta_info(void *arg, wlan_node_t node)
{
    ...
    if (wlan_node_getrssi(node, &rssi_info, WLAN_RSSI_RX) == 0) {
        si->isi_rssi = rssi_info.avg_rssi;
        si->isi_min_rssi = node->ni_rssi_min;
        si->isi_max_rssi = node->ni_rssi_max;
    }
    ...
}

```

To obtain the data/packets rate per client, an IC timer is started to record the data/packets during the defined duration.

To initialize a timer per IC to calculate the rate:

```

int ieee80211_ifattach (struct ieee80211com *ic, IEEE80211_REG_PARAMETERS
*ieee80211_reg_parm) {
    ...
+ifdef ATH_SUPPORT_EXT_STAT

```

```

+    //Initialize the timer when radio attached
+    OS_INIT_TIMER(ic->ic_osdev, &ic->ic_client_stat_timer, ieee80211_
client_stat_timeout, (void *) (ic));
+#endif
...
}

```

To start the timer when IC start running:

```

void ieee80211_start_running(struct ieee80211com *ic)
{
...
+
+#ifdef ATH_SUPPORT_EXT_STAT
+    //Start the timer when IC start running
+    OS_SET_TIMER (&ic->ic_client_stat_timer, CLIENT_STAT_UPDATE_
TIME*1000);
+#endif
...
}

```

To stop the timer when the IC stopped:

```

void ieee80211_stop_running(struct ieee80211com *ic)
{
...
+
+#ifdef ATH_SUPPORT_EXT_STAT
+    //Stop the timer when IC stopped
+    OS_CANCEL_TIMER (&ic->ic_client_stat_timer);
+#endif
    return 0;
}

```

To delete the timer when the IC detached:

```

void
ieee80211_ifdetach(struct ieee80211com *ic)
{
...
+
+#ifdef ATH_SUPPORT_EXT_STAT
+    //Free the timer when radio dettached
+    OS_FREE_TIMER (&ic->ic_client_stat_timer);
+#endif
...
}

```

To calculate the rate in timeout function:

```

+#ifdef ATH_SUPPORT_EXT_STAT
+static OS_TIMER_FUNC(ieee80211_client_stat_timeout)
+{
...
+
+    //Iterate all the associated stations per vap for the radio and call
cal_client_stat()
+
+    TAILQ_FOREACH(tmpvap, &ic->ic_vaps, iv_next) {
+
+        wlan_iterate_station_list(vap, cal_client_stat, &req);
+
+    }
+
+    //Restart the timer

```

```

+    OS_SET_TIMER (&ic->ic_client_stat_timer, CLIENT_STAT_UPDATE_
TIME*1000);
+
}
+endif

+ifdef ATH_SUPPORT_EXT_STAT
+static void cal_client_stat(void *arg, wlan_node_t node)
{
    ...
+    u_int32 rx_bytes_rate, rx_data_rate, tx_bytes_rate, tx_data_rate;
...
+    //Calculate the rate
+    rx_bytes_rate = (ni->ni_stats.ns_rx_bytes - ni->ni_stats.ns_rx_bytes_
last) / CLIENT_STAT_UPDATE_TIME;
+    rx_data_rate = (ni->ni_stats.ns_rx_data - ni->ni_stats.ns_rx_data_
last) / CLIENT_STAT_UPDATE_TIME;
+    tx_bytes_rate = (ni->ni_stats.ns_tx_bytes_success - ni->ni_stats.ns_
rx_bytes_success_last) / CLIENT_STAT_UPDATE_TIME;
+    tx_data_rate = (ni->ni_stats.ns_tx_data_success - ni->ni_stats.ns_rx_
data_success_last) / CLIENT_STAT_UPDATE_TIME;

+    //Update rx bytes per second
+    IEEE80211_NODE_STAT_SET(ni, rx_bytes_rate, rx_bytes_rate);
+    IEEE80211_NODE_STAT_SET(ni, rx_data_rate, rx_data_rate);
+    IEEE80211_NODE_STAT_SET(ni, tx_bytes_rate, tx_bytes_rate);
+    IEEE80211_NODE_STAT_SET(ni, tx_data_rate, tx_data_rate);

+    //Update statistics for last time
+    IEEE80211_NODE_STAT_SET(ni, rx_bytes_last, ni->ni_stats.ns_rx_bytes);
+    IEEE80211_NODE_STAT_SET(ni, rx_data_last, ni->ni_stats.ns_rx_data);
+    IEEE80211_NODE_STAT_SET(ni, tx_bytes_success_last, ni->ni_stats.ns_
tx_bytes_success);
+    IEEE80211_NODE_STAT_SET(ni, tx_data_success_last, ni->ni_stats.ns_tx_
data_success);
...
}
+endif

```

## 22.8.2 Sample output of apstats command

The following changes are made for “apstats -s -m” command to gather and show the statistics based on node:

```

static int nodelevel_gather_stats(int sockfd,
                                nodelevel_stats_t *nodestats)
{
    ...
+ifdef ATH_SUPPORT_EXT_STAT
+    nodestats->tx_stbc = si->tx_stbc;
+    nodestats->rx_stbc = si->rx_stbc;
+    nodestats->chwidth = si->chwidth;
+
+    nodestats->tx_bytes_rate = ns->ns_tx_bytes_rate;

```

```

+    nodestats->tx_data_rate = ns->ns_tx_data_rate;
+    nodestats->rx_bytes_rate = ns->ns_rx_bytes_rate;
+    nodestats->rx_data_rate = ns->ns_rx_data_rate;
+#endif
...
}
static int nodelevel_display_stats(nodelevel_stats_t *nodestats)
{
...
    APSTATS_PRINT_STAT8( "Rx RSSI" ,  nodestats->rx_rssi);
    APSTATS_PRINT_STAT8( "Rx MGMT RSSI" ,  nodestats->rx_mgmt_
rssi);
#define ATH_SUPPORT_EXT_STAT
+    display_band_width(nodestats->chwidth);
+    APSTATS_PRINT_GENERAL("stbc - tx(%d) rx(%d)\n", nodestats->tx_stbc,
nodestats->rx_stbc);
+    APSTATS_PRINT_GENERAL("chainmask (NSS) - tx: %d rx: %d \n", (si->isi_
nss & 0x0f), ((si->isi_nss >> 4) & 0x0f));
+    APSTATS_PRINT_STAT64("Rx byte per second" ,      nodestats->rx_bytes_
rate);
+    APSTATS_PRINT_STAT64("Rx packet per second " ,   nodestats->rx_data_
rate);
+    APSTATS_PRINT_STAT64("Tx byte per second" ,      nodestats->tx_bytes_
rate);
+    APSTATS_PRINT_STAT64("Tx packet per second " ,   nodestats->tx_data_
rate);
#endif

APSTATS_PRINT_GENERAL( "\n");
...
}
#endif ATH_SUPPORT_EXT_STAT
static void display_band_width(enum ieee80211_cwm_width ni_chwidth)
{
    switch(ni_chwidth)
    {
        case IEEE80211_CWM_WIDTH20:
            APSTATS_PRINT_GENERAL( "Band Width = 20\n" );
            return;
        case IEEE80211_CWM_WIDTH40:
            APSTATS_PRINT_GENERAL( "Band Width = 40\n" );
            return;
        case IEEE80211_CWM_WIDTH80:
            APSTATS_PRINT_GENERAL( "Band Width = 80\n" );
            return;
        case IEEE80211_CWM_WIDTH160:
        case IEEE80211_CWM_WIDTH80_80:
            APSTATS_PRINT_GENERAL( "Band Width = 160\n" );
            return;
    }
}
#endif

```

The following example shows the output of the apstats command with the newly introduced fields:

```

root@OpenWrt:/# apstats -s -m 00:03:7f:80:51:73
Node Level Stats: 00:03:7f:80:51:73 (under VAP ath0)
Tx Data Packets          = 0
Tx Data Bytes            = 0
Rx Data Packets          = 1
Rx Data Bytes            = 84
Tx Unicast Data Packets = 0
Average Tx Rate (kbps)   = 433300
Average Rx Rate (kbps)   = 433300
Last tx rate             = 433300
Last rx rate             = 433300
Last mgmt rx rate       = 6000
Rx MIC Errors           = 0
Rx Decryption errors    = 0
Rx errors                = 0
Packets Queued          = 0
Host Discard             = 0
Tx failures              = 0
Rx RSSI                  = 44
Rx MGMT RSSI             = 56
Band Width               = 80
stbc                     tx(0) rx(1)
chainmask (NSS)          tx(1) rx(1)
Rx byte per second       = 0
Rx packet per second     = 0
Tx byte per second       = 0
Tx packet per second     = 0

```

## 22.9 Connection state logging functionality

Connection State Logging (CSL) is print level debug frame work to aid CST to detect connection level issues automatically without human intervention. A user-configurable (enable/disable control for CSL) is introduced. Enter the `iwpriv ath0 csl <1/0>` command to enable (1) or disable (0) this logging mechanism. Enabling CSL should enable IEEE80211\_AUTH\_MSG and IEEE80211\_ASSOC\_MSG debug flags

CSL is supported for the following operations:

- Probe-related (Request and Response)
- Authentication-related (Request and Response)
- Association-related (Request and Response)
- Security-related (Handshake, Failure, Pass)

All WLAN frames involved in connection (probe requests and responses, authentication requests and responses, and association requests and responses) are logged.

## 22.10 Smart logging method

An event-based logging framework is introduced to reduce the effort and time consumed in reproducing problems or defects for test teams and customers by collecting all the necessary logs for various possible error conditions. These events can be scenarios such as copy engine failure, Tx/Rx not working, connection failure, and ping failure. The trigger-based logging or smart logging framework (also called scenario-based logging) is responsible for the following:

- Identification of the trigger conditions
- Collection of necessary debug information, for the trigger condition, by interacting with Host/FW
- Flushing of any residual log (in the internal log buffers) in the Host/FW
- Logging the collected information in persistent memory
- Handle logging without performance impact when all log levels (control path) are enabled in Host/FW

Starting with the QCA\_Networking\_2017.SP5.0 release, the following changes are implemented:

- Identification of the trigger conditions
- Collection of necessary debug information, for the trigger condition, by interacting with host/FW
- Flushing of any residual log (in the internal log buffers) in the host/FW

This mechanism ensures that the necessary debug information is available at least on the console output for the identified error scenarios.

### 22.10.1 Host-FW interaction

Host and FW need to interact with each other for the following purposes:

- Host to indicate FW regarding fatal/trigger scenarios
- FW to indicate Host regarding fatal/trigger scenarios
- FW to communicate the scenario-based additional logs

#### 22.10.1.1 Indication of fatal event occurs through WMI messaging

In this method, the fatal events are sent using debug event WMI messages between host and firmware (FW). Host and FW are capable of sending debug events to each other, they have ability to parse these messages and handle them as necessary. This method is bulky and causes an additional WMI message and handling interface.

To mitigate the risk, an analysis was conducted to use HTT message through packetlog copy engine (CE) instead of WMI CE. Because the packetlog CE is used only during throughput debug purpose, the same copy engine can be used to indicate the fatal events. However, CE8 is used only for target to host communication (PIPEDIR\_IN). So, CE8 has to be made bidirectional (PIPEDIR\_INOUT) for this smart logging purpose which is not possible. Therefore, CE8 cannot be used.

The regular debug messages from FW are coming through WMI CE. If the smart logging-related messages are coming through a separate CE, these messages can be out of synchronization with relevance to the timestamps between the copy engines. Host or userspace needs to sort the debug messages based on timestamp.

Instead of bringing in a new messaging format over these CEs which are being used for other purpose, the existing WMI CE is reused for smart logging.

If the WMI CE is going to be used for fatal event indication, it has to be used with caution. If there are false detections, it can lead to performance impact. If the WMI CE itself is down, no further communication can happen. However, this is an expected behavior because the system transitions into target-assert state after watchdog alert.

### 22.10.1.2 WMI message from host to FW

**WMI\_DEBUG\_FATAL\_CONDITION\_CMDID:** Host uses this WMI command to indicate any fatal event to FW. After the FW receives this command, it performs the following:

- Log any information needed to debug this fatal condition: These are standard messages carried over the existing WMI message ‘WMI\_DEBUG\_MESG\_EVENTID’ Flush any pending logs in the FW buffer: These flushed logs would also be carried just as they are being done today using ‘WMI\_DEBUG\_MESG\_EVENTID’
- Send a completion event to the Host

### 22.10.1.3 WMI message from FW to host

**WMI\_DEBUG\_FATAL\_CONDITION\_EVENTID:** FW sends this event in the following two scenarios:

- FW detecting any fatal scenario: After detecting such a scenario, FW is expected to log any necessary information and flush any pending logs, if any.
- Response to WMI\_DEBUG\_FATAL\_CONDITION\_CMDID

After the host receives this event, it performs the following:

- Log any information needed to debug this fatal condition. This is needed in case of indication only. Not needed for completion since Host was the one that originally identified the error.
- Log the received logs in persistent memory.

### 22.10.1.4 WMI message format

The following is the message data structure:

```
typedef struct {
    A_UINT32 type;
    A_UINT32 subtype;
    A_UINT32 reserved0;
} wmi_fatal_condition;
typedef struct {
    A_UINT32 num_events;
    // Followed by num_events * wmi_fatal_condition
```

```
}
```

**Table 22-1 Type and subtypes**

| Event                  | Type | Sub Event    | Subtype |
|------------------------|------|--------------|---------|
| Fatal event completion | 0x00 | NA           | NA      |
| CE debug event         | 0x01 | NA           | NA      |
| Timeouts               | 0x02 | TX timeouts  | 0x00    |
|                        |      | RX timeouts  | 0x01    |
| Connection issue       | 0x03 | STA kick out | 0x00    |

- The same message format can be used for both WMI\_DEBUG\_FATAL\_CONDITION\_CMDID and WMI\_DEBUG\_FATAL\_CONDITION\_EVENTID.
- Type 0x00 is used by FW to send completion event to the Host i.e., this will be sent in response to WMI\_DEBUG\_FATAL\_CONDITION\_CMDID from host
- All other fatal indications from Host/FW contain a unique type-subtype combination.
- num\_events helps in cases where logs of more than one fatal event are required in the future. Currently, this value is 1.

### 22.10.1.5 Enable smart logging using WMI

A new WMI command (WMI\_CONFIG\_SMART\_LOGGING\_CMDID) is introduced to enable/disable this smart logging feature.

The message is of the format:

```
typedef struct {
    A_UINT32 enable;
} wmi_smart_logging;
```

The scenario-based logs continue to use the WMI\_DEBUG\_MESG\_EVENTID itself.

### 22.10.2 Identification of trigger conditions

From Host, identification of trigger conditions is not implemented. Identification of trigger conditions from FW is implemented for CE-full conditions, STA kickouts, and Tx/Rx timeouts.

#### 22.10.2.1 CE-full condition

Currently in FW, when CE failure happens, retry occurs  $2^{24}$  times. Before this failure, watchdog alert is triggered and the read/write indexes are updated. Therefore, after the retry count exceeds 100, for every 100 retries, FW starts storing the corresponding read/write index of the CE in the global variable '**g\_patch\_dbg\_CE\_debug\_data**'. Apart from storing these indexes, the FW sends

an indication to the host. When a crash happens due to CE stuck, this variable is examined to determine the read/write index and understand whether the packets were reaped or not. This indication to the host is currently available only for CE5. Problems with CE2 cannot be communicated because it uses WMI message and if the WMI CE itself is down, only the dumps and logged read/write index are available.

### 22.10.2.2 Tx/Rx timeouts

The monitor task in FW runs for every 200 ms. This task is updated to accommodate the monitoring of Tx and Rx timeouts. For every 20 Tx/Rx failures, an indication is sent to the host. The following registers are printed:

- MAC\_PCU\_RX\_CLEAR\_CNT\_ADDRESS and MAC\_PCU\_CYCLE\_CNT\_ADDRESS—Enable a user to identify whether the medium is busy
- PHY\_BB\_BBB\_DAGC\_CTRL\_ADDRESS, PHY\_BB\_FIND\_SIGNAL\_ADDRESS, and PHY\_BB\_TIMING\_CONTROL\_5\_ADDRESS—Enable a user read ANI-related registers
- Noise floor value – Primary and Secondary
- PHY\_BB\_AGC\_CONTROL\_ADDRESS and PHY\_BB\_AGC\_CONTROL\_DO\_NOISEFLOOR\_MASK—Enable a user to identify whether the noise floor calibration is successful or not

### 22.10.2.3 STA kickouts

The following parameters are printed at 80% and 90% of the STA kickout thresholds. By the time the peer is disconnected, a snapshot of the system condition with the following parameters can be obtained:

- The registers previously mentioned in this section
- Rx timeout count, Rx sequence mismatch count, Rx watchdog timeout
- Tx timeout count

## 22.11 CE logging functionality

The copy engine (CE) logging capability enables logging of information relating to CE descriptors and associated buffers. In addition, this functionality also supports framework for Smart Logging in which fatal events from FW are recorded or dumped.

This CE logging functionality provides a mechanism to enable and disable the Smart Logging feature. A user can collect and store the history of CE descriptors and fixed ‘N’ bytes of associated buffers. Also, FW fatal event messages are recorded when the Smart Logging is enabled.

The sysfs file interface is used for dumping FW fatal logs and current CE descriptors and associated data buffers. In addition, the sysfs file interface is used for dumping CE descriptors and fixed ‘N’ bytes of associated data buffers.

For CE debugging, user can save all the history of CE descriptors and the fixed n bytes of the buffer pointed by it on to the console. For Smart logging, the FW logs and current CE descriptors with N bytes of data buffer pointed by it can be dumped by the user on the console.

The FW logs are stored in a Smart log memory area, whenever a Smart Log event comes from the FW, and can be dumped on the console by the user.

### 22.11.1 Storage space of logs

1. The FW logs and the copy engine (CE) dumps are stored in a storage space of 64 pages of allocated valloc.
2. The minimum space required for each CE dump is as follows:
  - a. Size of CE Ring state = 60 bytes
  - b. Size of each CE descriptor = 8 bytes
  - c. nentries is the number of CE descriptors for each CE.
  - d. Total size = a + (b\*c)
3. For QCA9984/QCA9980, the storage space is calculated as follows:
  - a. nentries for each CE:
 

CE\_id0 = 16, CE\_id1 = 0, CE\_id2 = 0, CE\_id3 = 32, CE\_id4 = 4096,  
 CE\_id5 = 0, CE\_id6 = 0, CE\_id7 = 2, CE\_id8 = 0, CE\_id9 = 0,  
 CE\_id10 = 0, CE\_id11 = 0
  - b. Total nentries (16 + 32 + 4096 + 2)
  - c. Size of each CE descriptor = 8 bytes
  - d. Size of CE ring state for all CEs = 60 bytes \* 12CEs = 720 bytes.
  - e. Total space required = b\*c + d = 33888 = 8Pages (4096/page)
4. Also, the skb pointed by each CE descriptor is dumped, which is user-defined using iwpriv commands.

### 22.11.2 Storage format of logs

- When a FW log arrives from the FW in an encoded format it is directly memory-copied into the storage space pointed by the current pointer and the current pointer is advanced by length of the FW log. The encoded log is appended with FWL ascii value, which functions as a delimiter while parsing the logs.
- Similarly, the various logs stored in the storage file are appended with the respective log types:
  - Firmware debug logs: Append FWL
  - Firmware Smartlog events: Append EVT
  - CE register dumps : Append CE<CE\_id>

- Before copying the encoded logs in to the storage area, the available space is checked and if the space available is less than the required space, then the logs are copied from the beginning of the buffer.

### 22.11.3 Retrieval of logs

The sysfs is used for the dumping of all the logs on to the console. The following two sysfs entries exist:

- For dumping all smart logs that contain FW logs and smart log event dumps (CE dumps), enter the \$cat /sys/class/net/wifi0/smartlogs\_dump command.
- For dumping history of CE descriptors and the fixed n bytes of the buffer pointed by it on to the console, enter the \$cat /sys/class/net/wifi0/celogs\_dump command.

### 22.11.4 Usage information

For dumping all smart logs that contain FW logs and SmartLog events, set the SKB size of the associated buffer that needs to be stored by entering the following commands:

```
$iwpriv wifiX smartLogSkbSz <size>
$cat /sys/class/net/wifiX/smartlogs_dump
```

For dumping history of CE descriptors and the fixed 1N1 bytes of buffer pointed by it on console, enable CE debug logs and dump the CE buffer history on the console by entering the following commands:

```
$iwpriv wifiX ce_debug_en <1/0>
$cat /sys/class/net/wifiX/celogs_dump
```

### 22.11.5 Examples of CE logs

The following is an example of CE logs:

```
root@OpenWrt:/# cat /sys/class/net/wifi0/celogs_dump root@OpenWrt:/# cat
/sys/class/net/wifi1/celogs_dump
[ 827.520024] -----CE:0-----
[ 827.527425] Buffer1
[ 827.529400]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.534955]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.540585]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.546206]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.551869]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.555493] Buffer2
[ 827.557468]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.563145]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.568794]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.574495]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.580183]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.583592] Buffer3
[ 827.585679]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.597599]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
[ 827.602296]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.608047]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.613571]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.617132] Buffer4
[ 827.619318]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.624849]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.630592]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.636127]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ 827.641870]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

## 22.11.6 Examples of smart logs

The following is an example of partial smart logs:

```
root@OpenWrt:/# root@OpenWrt:/#
root@OpenWrt:/# iwpriv wifi0 | grep smt
smtLgEvent(115E) : set1 int& get0 smtLgSkbsz(1160) : set1 int& get0
g_smtLgSkbsz(1160) : set0& get1 int root@OpenWrt:/# iwpriv wifi0 smtLgSkbsz 10

root@OpenWrt:/# iwpriv wifi0 smtLgEvent 1
[13704.506609] [wifi0] FWLOG: [60476] UNKNOWN 22:55 ( 0x1, 0x0, 0xab22 )
[13704.512018] SmartLogEvent: CE_DEBUG_EVENT

root@OpenWrt:/# [13704.517394] Cannot print Ring for Unused CEO [13704.521962]
Cannot print Ring for Unused CEO
[13704.525901] Cannot print Ring for Unused CEO [13704.530319] Cannot print Ring
for Unused CEO

root@OpenWrt:/# root@OpenWrt:/#
root@OpenWrt:/# cat /sys/class/net/wifi0/smartlogs_dump

[13723.757865]e0201000:46574c 4e 2a 00 00 2e 58 fc 17
03 00 00 00 1e FWLN*...X.....
[13723.765199]e0201010:000000 4c 09 00 00 4c 09 00 00
00 00 00 00 4e ...L...L.....N
[13723.773269]e0201020:2a0000 2e 58 fc 17 12 00 00 00
1e 00 00 00 4c *...X.....L
[13723.781354]e0201030:090000 4c 09 00 00 00 00 00 00
4e 2a 00 00 2e ...L.....N*...
[13723.789257]e0201040:58fc17 45 00 00 00 1e 00 00 00
4c 09 00 00 4c X..E.....L...
[13723.797453]e0201050:090000 00 00 00 00 4e 2a 00 00
2e 58 fc 17 67 .....N*...X..g
[13723.805697]e0201060:000000 1e 00 00 00 4c 09 00 00
4c 09 00 00 00 .....L...L...
[13723.813494]e0201070:000000 46 57 4c 7e 2b 00 00 37
58 fc 17 3a 00 ...FWL~+..7X...
[13723.827840]e0201080:000010 0a 00 00 60 01 00 00 10
00 00 00 00 ..... .
[13723.834894]e0201090:00007e 2b 00 00 37 58 fc 17 3a
00 00 00 10 00 ..~+..7X...:....
[13723.843153]e02010a0:000000 00 00 00 10 00 00 00 12
00 00 00 46 57 .....FW
[13723.851188]e02010b0:4c3245 00 00 37 58 fc 17 16 00
00 00 10 0a 00 L2E..7X.....
```

|                               |    |    |    |    |                  |         |    |    |    |
|-------------------------------|----|----|----|----|------------------|---------|----|----|----|
| [13723.859252]e02010c0:001008 | 00 | 00 | 10 | 00 | 00               | 00      | 00 | 00 | 00 |
| 00                            | 00 | 32 | 45 | 00 | .....            | 2E.     |    |    |    |
| [13723.867185]e02010d0:003758 | fc | 17 | 16 | 00 | 00               | 00      | 10 | 00 |    |
| 00                            | 00 | 00 | 00 | 00 | .7X.....         |         |    |    |    |
| [13723.875423]e02010e0:001000 | 00 | 00 | 12 | 00 | 00               | 00      | 46 | 57 |    |
| 4c                            | 7f | 47 | 00 | 00 | .....            | FWL.G.. |    |    |    |
| [13723.883491]e02010f0:0558fc | 0b | 3c | 14 | 00 | 00               | 00      | 00 | 00 |    |
| 00                            | b9 | 48 | 00 | 00 | .X..<.....H..    |         |    |    |    |
| [13723.891542]e0201100:0558fc | 0b | 50 | 14 | 00 | 00               | 00      | 00 | 00 |    |
| 00                            | f3 | 49 | 00 | 00 | .X..P.I..        |         |    |    |    |
| [13723.899618]e0201110:0558fc | 0b | 64 | 14 | 00 | 00               | 00      | 00 | 00 |    |
| 00                            | 2d | 4b | 00 | 00 | .X..d.....-K..   |         |    |    |    |
| [13723.907575]e0201120:0558fc | 0b | 78 | 14 | 00 | 00               | 00      | 00 | 00 |    |
| 00                            | 46 | 57 | 4c | 67 | .X..x.....       | FWLg    |    |    |    |
| [13723.915698]e0201130:4c0000 | 05 | 58 | fc | 0b | 8c               | 14      | 00 | 00 |    |
| 00                            | 00 | 00 | 00 | a0 | L...X.....       |         |    |    |    |
| [13723.923783]e0201140:4d0000 | 05 | 58 | fc | 0b | a0               | 14      | 00 | 00 |    |
| 00                            | 00 | 00 | 00 | db | M...X.....       |         |    |    |    |
| [13723.931846]e0201150:4e0000 | 05 | 58 | fc | 0b | b4               | 14      | 00 | 00 |    |
| 00                            | 00 | 00 | 00 | 46 | N...X.....       | F       |    |    |    |
| [13723.939908]e0201160:574c15 | 50 | 00 | 00 | 05 | 58               | fc      | 0b | c8 |    |
| 14                            | 00 | 00 | 00 | 00 | WL.P...X.....    |         |    |    |    |
| [13723.947998]e0201170:00004e | 51 | 00 | 00 | 05 | 58               | fc      | 0b | 7c |    |
| 15                            | 00 | 00 | 00 | 00 | ..NQ...X.. ....  |         |    |    |    |
| [13723.955988]e0201180:000088 | 52 | 00 | 00 | 05 | 58               | fc      | 0b | 90 |    |
| 15                            | 00 | 00 | 00 | 00 | ..R...X.....     |         |    |    |    |
| [13723.964148]e0201190:000046 | 57 | 4c | c3 | 53 | 00               | 00      | 05 | 58 |    |
| fc                            | 0b | a4 | 15 | 00 | ..FWL.S...X..... |         |    |    |    |
| [13723.972212]e02011a0:000000 | 00 | 00 | fd | 54 | 00               | 00      | 05 | 58 |    |
| fc                            | 0b | b8 | 15 | 00 | .....T...X.....  |         |    |    |    |
| [13723.980266]e02011b0:000000 | 00 | 00 | 37 | 56 | 00               | 00      | 05 | 58 |    |
| fc                            | 0b | cc | 15 | 00 | ....7V...X.....  |         |    |    |    |
| [13723.988359]e02011c0:000000 | 00 | 00 | 71 | 57 | 00               | 00      | 05 | 58 |    |
| fc                            | 0b | e0 | 15 | 00 | ....qW...X.....  |         |    |    |    |
| [13723.996353]e02011d0:000000 | 00 | 00 | 46 | 57 | 4c               | aa      | 58 | 00 |    |
| 00                            | 05 | 58 | fc | 0b | ....FWL.X...X..  |         |    |    |    |
| [13724.004540]e02011e0:f41500 | 00 | 00 | 00 | 00 | 00               | e5      | 59 | 00 |    |
| 00                            | 05 | 58 | fc | 0b | .....Y...X..     |         |    |    |    |
| [13724.012580]e02011f0:081600 | 00 | 00 | 00 | 00 | 00               | 1f      | 5b | 00 |    |
| 00                            | 05 | 58 | fc | 0b | .....[...X..     |         |    |    |    |
| [13724.020662]e0201200:1c1600 | 00 | 00 | 00 | 00 | 00               | 46      | 57 | 4c |    |
| 59                            | 5c | 00 | 00 | 05 | .....FWLY\...    |         |    |    |    |
| <br>                          |    |    |    |    |                  |         |    |    |    |
| [13724.028761]e0201210:58fc0b | 30 | 16 | 00 | 00 | 00               | 00      | 00 | 00 |    |
| 93                            | 5d | 00 | 00 | 05 | X..0.....]...    |         |    |    |    |
| [13724.036715]e0201220:58fc0b | 44 | 16 | 00 | 00 | 00               | 00      | 00 | 00 |    |
| cc                            | 5e | 00 | 00 | 05 | X..D.....^...    |         |    |    |    |
| [13724.044872]e0201230:58fc0b | 71 | 16 | 00 | 00 | 00               | 00      | 00 | 00 |    |
| 46                            | 57 | 4c | 07 | 60 | X..q.....FWL..   |         |    |    |    |
| [13724.052931]e0201240:000005 | 58 | fc | 0b | 85 | 16               | 00      | 00 | 00 |    |
| 00                            | 00 | 00 | 41 | 61 | ...X.....Aa      |         |    |    |    |
| [13724.061022]e0201250:000005 | 58 | fc | 0b | 99 | 16               | 00      | 00 | 00 |    |
| 00                            | 00 | 00 | 7b | 62 | ...X.....{b      |         |    |    |    |
| [13724.069082]e0201260:000005 | 58 | fc | 0b | ad | 16               | 00      | 00 | 00 |    |
| 00                            | 00 | 00 | 46 | 57 | ...X.....FW      |         |    |    |    |

|                               |    |    |    |    |                 |    |    |    |
|-------------------------------|----|----|----|----|-----------------|----|----|----|
| [13724.077082]e0201270:4cb563 | 00 | 00 | 05 | 58 | fc              | 0b | c1 | 16 |
| 00                            | 00 | 00 | 00 | 00 | L.c....X.....   |    |    |    |
| [13724.085233]e0201280:00f364 | 00 | 00 | 23 | 30 | fc              | 07 | 00 | 00 |
| 00                            | 00 | f3 | 64 | 00 | ..d..#0.....d.  |    |    |    |
| [13724.093291]e0201290:002330 | fc | 07 | 00 | 00 | 00              | 00 | f8 | 64 |
| 00                            | 00 | 09 | 38 | 00 | .#0.....d...8.  |    |    |    |
| [13724.101384]e02012a0:100816 | 00 | 00 | 02 | 00 | 00              | 00 | 00 | 00 |
| 00                            | 00 | 01 | 00 | 00 | .....           |    |    |    |
| [13724.109445]e02012b0:00f864 | 00 | 00 | 05 | 58 | fc              | 0b | 08 | 16 |
| 0f                            | 00 | 00 | 00 | 00 | ..d....X.....   |    |    |    |
| [13724.117445]e02012c0:003767 | 00 | 00 | 14 | 38 | fc              | 07 | 00 | 00 |
| 00                            | 00 | 46 | 57 | 4c | .7g...8.....FWL |    |    |    |
| [13724.125594]e02012d0:3cec00 | 00 | 37 | 58 | fc | 0f              | 01 | 00 | 00 |
| 00                            | 00 | 00 | 00 | 00 | <...7X.....     |    |    |    |
| [13724.133657]e02012e0:22ab00 | 00 | 43 | 45 | 30 | 10              | 00 | 00 | 00 |
| 0f                            | 00 | 00 | 00 | 02 | "...CEO.....    |    |    |    |
| [13724.141746]e02012f0:000000 | 02 | 00 | 00 | 00 | 02              | 00 | 00 | 00 |
| 00                            | 80 | ff | de | 00 | .....           |    |    |    |
| [13724.149811]e0201300:c0525b | 00 | 80 | ff | de | 00              | c0 | 52 | 5b |
| 80                            | 2a | 7f | d8 | 80 | .R[.....R[.*... |    |    |    |

## 22.12 Buffer tracking

The capability to display the count of the active skbuffs being used by driver at particular instance to help in identifying any leak of buffers is supported. A global counter is maintained, which is incremented after allocation and decremented after skb is freed. The difference is used to analyze any active memory leaks.

This feature keeps track of the network buffers owned by Wi-Fi driver by maintaining a global count variable, which is incremented at allocation and decremented at free. This variable is also modified to contain the ownership of buffer changes (that is, when buffer is handed over to Linux or networking subsystem [NSS], the counter decrements and increments when buffer is handed to Wi-Fi driver from NSS or Linux). This method helps in the categorizing OOM problem. If count is high, it is possible that a problem exists with the Wi-Fi driver. Alternatively, if the count is less than 15000, the memory in Wi-Fi driver is stable. This feature can be enabled at compilation time by enabling the QDF\_NBUF\_GLOBAL\_COUNT macro. The command to view the counters is ‘cat /proc/net/qdf/nbuf\_counters’.

```
$ cat /proc/net/qdf/nbuf_counters
NBUF ACTIVE COUNT --> 6812
```

## 22.13 Packet log and Tx completion message correlation to mirror Tx data packets on QCA9880

On QCA9880, packet logs and Tx Completion Indication messages carry their own header. Because no correlation exists between them, it is difficult to perform post-processing of these two messages to obtain a unified Tx status. To overcome this problem, a methodology is implemented to correlate Packet Log Message and Tx Completion Indication message for the following cases:

- Host generated Tx data.

- Firmware/target generated Tx data (such as ADBA and DELBA)

Enhancements are made for packet log messages and Tx Completion Indication headers to carry a unique ID for a given frame (PPDU). Using this unique ID, host or a post processing application will be able to correlate them get a unified status.

The following capabilities are implemented:

- Provision of runtime flag to enable or disable this feature (Tx\_DATA\_CAPTURE)
- Definition of Unique ID for Packet Log and Tx Completion Indication messages
- Extension Tx Completion Indication to include unique ID
- Extension of Tx Management msg completion Indication to include unique ID
- Extension of Packet Log messages to include unique ID
- Definition of combined event for PPDU\_CTRL and PPDU\_STAT
- Definition of new event for local and self-generated event types for in Packet Log message

A new packet log event WMI\_PKTLOG\_EVENT\_Tx\_DATA\_CAPTURE (0x100) is defined to enable Tx mirroring feature. When this flag is set, the following event types are sent in the packet log:

- WAL\_PKT\_INFO\_TYPE\_Tx\_STATS\_COMBINED : combining PPDU\_STAT and PPDU\_CTRL
- WAL\_PKT\_INFO\_TYPE\_Tx\_LOCAL : sending local generated pkt stats
- WAL\_PKT\_INFO\_TYPE\_Tx\_SELFGEN : sending self-generated pkt stats

The following module is modified to set process and set the WMI command:

| Module        | Change description                                                                                                                                                                                                                |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| wmi_unified.h | enum <i>WMI_PKTLOG_EVENT</i><br>A new packet log event is defined for Tx capture.<br>WMI_PKTLOG_EVENT_Tx_DATA_CAPTURE = 0x100                                                                                                     |
| pktlog.h      | The following new types need to be added<br><code>#define PKTLOG_TYPE_Tx_STATS_COMBINED 11</code><br><code>#define PKTLOG_TYPE_Tx_LOCAL 12</code><br><code>#define PKTLOG_TYPE_Tx_SELFGEN 13</code>                               |
| wal_phy_dev.h | enum wal_pkt_info_type<br>WAL_PKT_INFO_TYPE_Tx_STATS_COMBINED<br>WAL_PKT_INFO_TYPE_Tx_LOCAL<br>WAL_PKT_INFO_TYPE_Tx_SELFGEN                                                                                                       |
| pktlog.c      | <b><i>pktlog_enable()</i></b><br>This function is modified to include event list for Tx_DATA_CAPTURE and the following types<br>WAL_PKT_INFO_TYPE_Tx_STATS_COMBINED<br>WAL_PKT_INFO_TYPE_Tx_LOCAL<br>WAL_PKT_INFO_TYPE_Tx_SELFGEN |

### 22.13.1 Unique ID for Tx Completion Indication

Because QCA9880 does not have schedule ID that is present in QCA9980 (In QCA9980, Packet Log already has type\_specific\_field which is filled with isr\_status->sched\_id. This ID is used in Tx Completion Indication as unique ID), a simple approach is used where a global counter is maintained as part of Tx context data structure (*wal\_tx\_ctxt*) and updated when new PPDU is queued to HW. This global counter is used as the unique ID for correlating Packet Log and Tx Completion Indication messages. Because QCA9880 is single-threaded, Packet Log and Tx Completion Indication are generated one after the other. Therefore, a common global counter (part of tx context structure) is adequate for correlating Packet Log and Tx Completion.

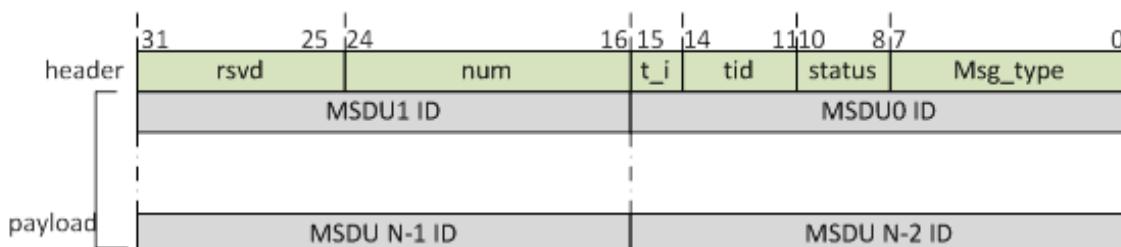
The following modules and structures are modified to incorporate these changes.

| Module           | Change description                                                                                                                                                                                                     |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ar_wal_tx_ds.h   | wal_tx_ctxt<br>Add new 16-bit field <i>ppdu_unique_Id</i> in <i>wal_tx_ctxt</i> data structure to store unique ID/seq number associated with each Tx frame (PPDU). This number is incremented when Tx done is handled. |
| ar_wal_tx_send.c | tx_send_proc_ppdu_done()<br>Modify this function to increment <i>ppdu_unique_Id</i>                                                                                                                                    |

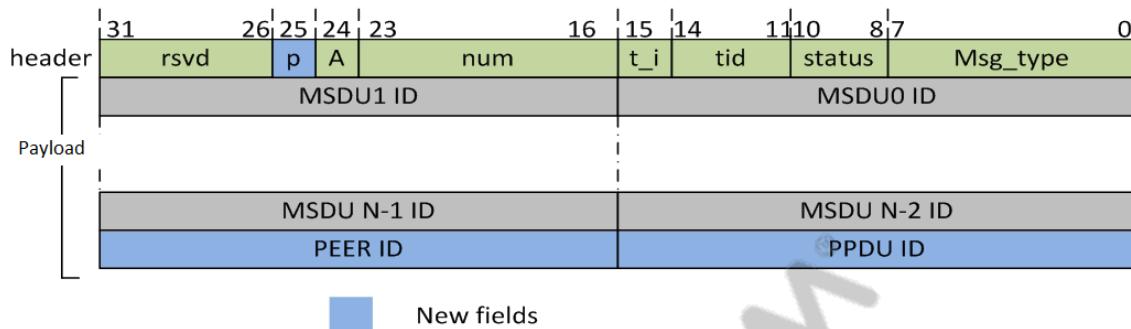
### 22.13.2 Tx Completion Indication

This message provides status to the host for the last transmitted frame (PPDU). This indication is generated as part of Tx done completion ISR received from HW. The HW provides status whether the PPDU transmission OK, NO\_ACK or DISCARDED.

#### Previous Tx Completion Indication packet format



### Modified Tx Indication packet format



#### Definition of new fields

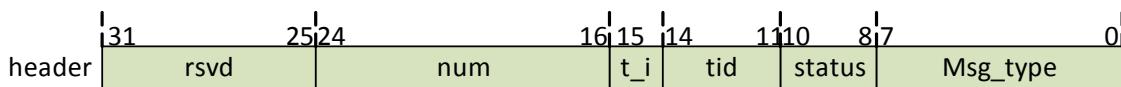
| Field   | Size(bits) | Description                                                    | Header/payload |
|---------|------------|----------------------------------------------------------------|----------------|
| P       | 1          | defines whether PEER ID and PPDU ID are present in the payload | Header         |
| PPDU ID | 16         | Uniquely identifies Tx frame                                   | Payload        |
| PEER ID | 16         | Uniquely associates Tx frame                                   | payload        |

PPDU ID is filled with the unique ID generated when processing Tx completion event from the HW.

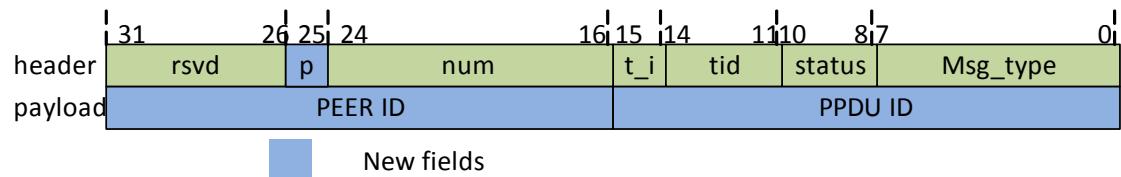
| Module       | Change description                                                                                    |
|--------------|-------------------------------------------------------------------------------------------------------|
| htt_tgt_tx.c | htt_tgt_tx_compl_ind()<br>This function is modified to include unique ID/Seq number in PPDU ID field. |

### 22.13.3 Management Tx Completion Indication

#### Previous Management Tx Indication packet format



#### Modified Management Tx Indication packet format



## Definition of new fields

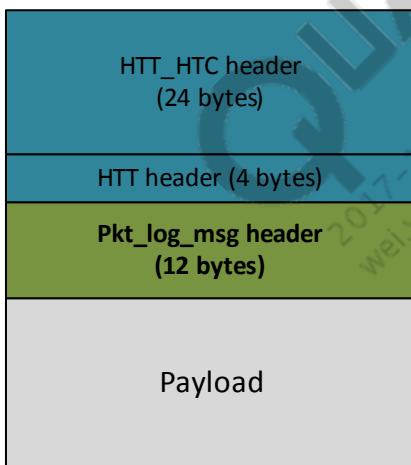
| Field   | Size(bits) | Description                                                    | Header/payload |
|---------|------------|----------------------------------------------------------------|----------------|
| P       | 1          | defines whether PEER ID and PPDU ID are present in the payload | Header         |
| PPDU ID | 16         | Uniquely identifies Tx frame                                   | Payload        |
| PEER ID | 16         | Uniquely associates Tx frame                                   | payload        |

PPDU ID is filled with the unique ID generated while processing Tx completion event from the HW.

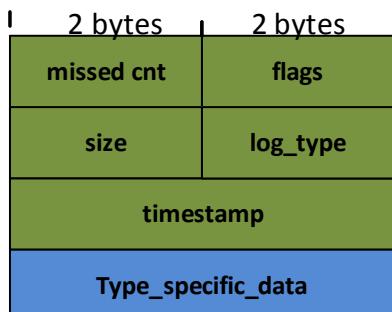
| Module       | Change description                                                                                         |
|--------------|------------------------------------------------------------------------------------------------------------|
| htt_tgt_tx.c | htt_tgt_mgmt_tx_compl_ind()<br>This function is modified to include unique ID/Seq number in PPDU ID field. |

### 22.13.4 Packet log message

The following diagram shows the existing packet log message format:



A new 4-byte field (type\_specific\_data) is added in pktlog\_hdr header portion of the message to send unique ID/sequence number associated with the message.



New fields

The following modules are modified to incorporate these changes:

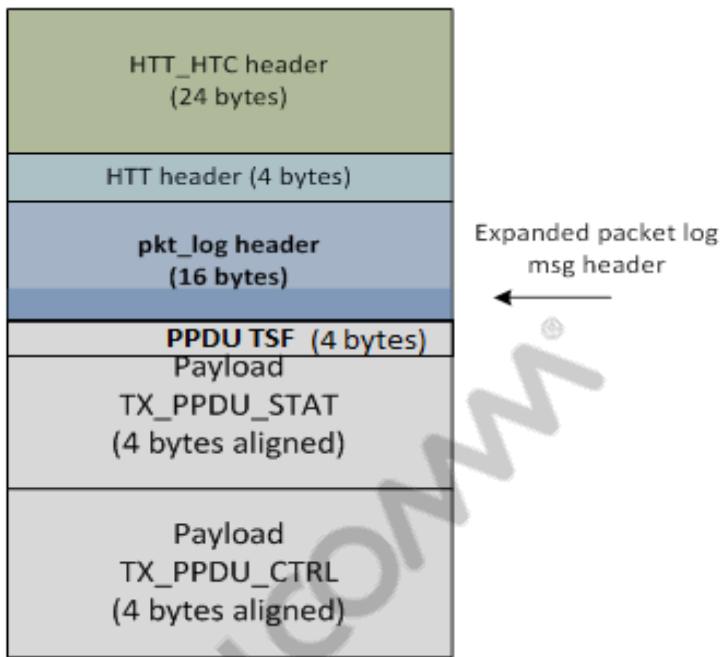
| Module   | Change description                                                                                                     |
|----------|------------------------------------------------------------------------------------------------------------------------|
| pktlog.h | pktlog_hdr<br>This structure is modified to include unique ID                                                          |
| pktlog.c | pktlog_sendbuf()<br>This function is modified to include <i>ppdu_unique_Id</i> in the <i>pkt_log_msg</i> header field. |

#### 22.13.4.1 Combined Tx\_PPDU\_STAT and Tx\_PPDU\_CTRL

In the default QCA9880 implementation, Tx\_PPDU\_STAT and Tx\_PPDU\_CTRL are sent as separate types. In this FR, these two event types are combined to produce a unified type Tx\_STATS\_COMBINED.

This event/type has been define in addition to the existing types Tx\_PPDU\_STAT and Tx\_PPDU\_CTRL. If Tx\_DATA\_CAPTURE event is enabled, then Tx\_STATS\_COMBINED type will be sent.

The following is the new packet format:



| Module           | Change description                                                                                                                           |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| ar_wal_tx_send.c | tx_send_pkt_info_tx_ppdu_stat_ppdu_ctrl_combined()<br>New method is defined to combine Tx_PPDU_STAT and Tx_PPDU_CTRL for packet log messages |

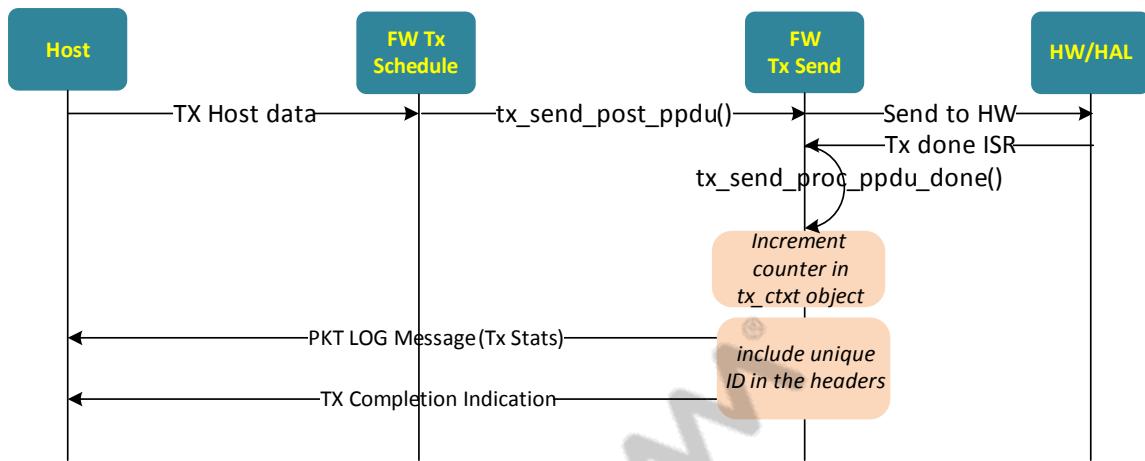
#### 22.13.4.2 Self-generated and locally-generated types

The following event types are defined for sending self-generated and locally-generated packet events to host:

| Module           | Change description                                                                                             |
|------------------|----------------------------------------------------------------------------------------------------------------|
| ar_wal_tx_send.c | tx_send_pkt_info_do_cb_local_self()<br>New method is defined to send self and locate generated PPDU Tx events. |

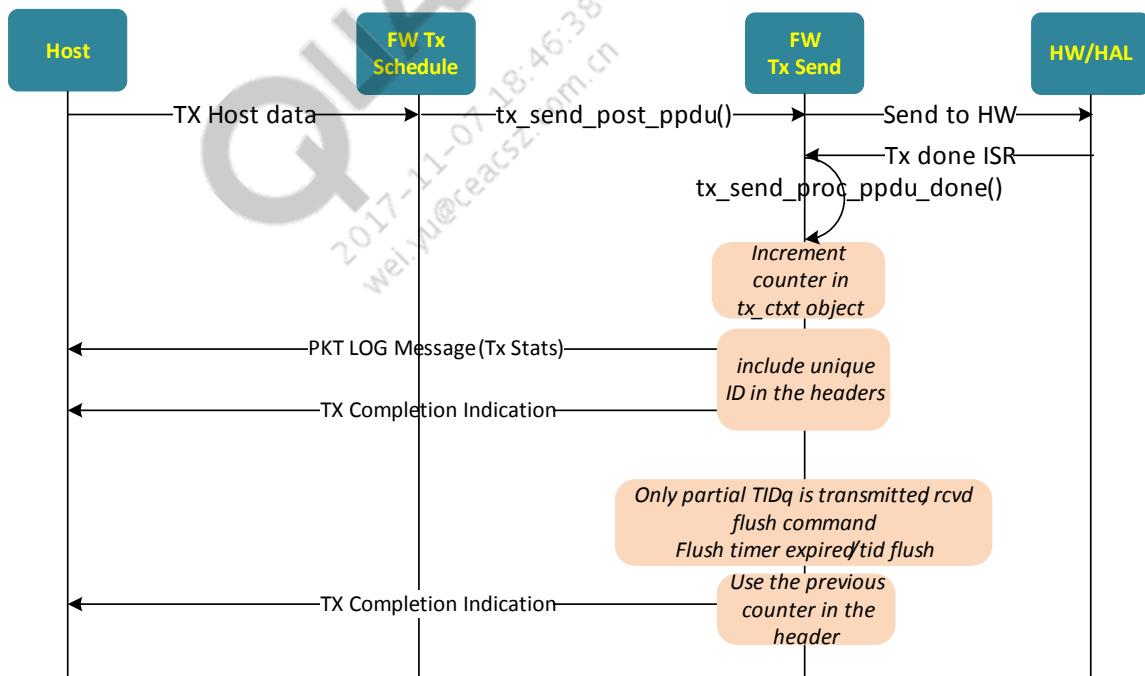
#### 22.13.5 Call flows

##### 22.13.5.1 Tx packet normal case

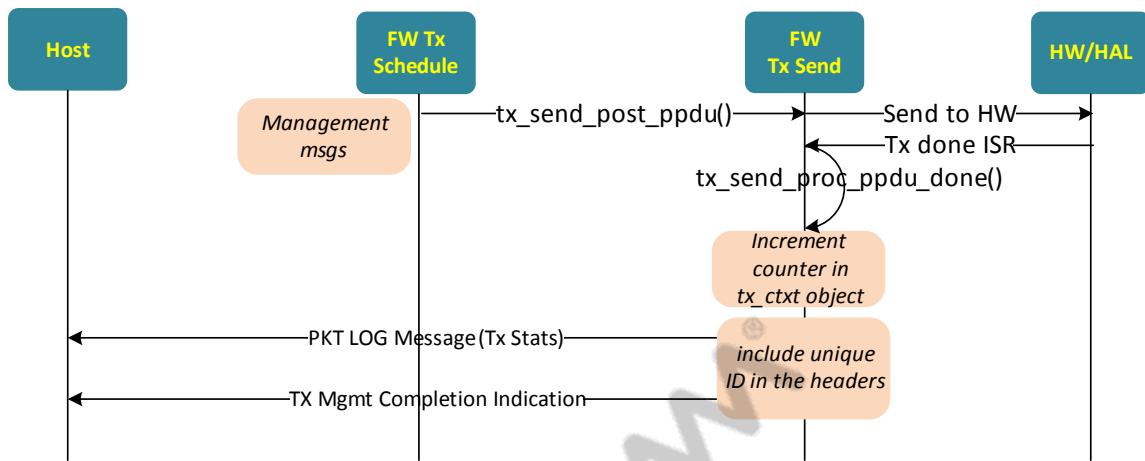


This design assumes there is no outstanding Tx done indication before initiating a new frame-send command to HW.

### 22.13.5.2 Flush expired/TID flush



### 22.13.5.3 Self and locally generated packets



## 22.14 Enhanced client statistics display for Tx management frames, minimum/maximum RSSI, and received data MPDUs

This section describes the display of enhanced statistics of connected clients, such as the number of frames at the entry of the UMAC layer, the number of Tx errors and retries for management frames, minimum and maximum RSSI, and the data MPDUs received from the radio. Use the `apstats -s -m xx:xx:xx:xx:xx:xx` (MAC address of station) command to view these enhanced client or STA statistical details. The following additional statistics are displayed:

| Type                  | Description                                                                                                                                                                                                                                    |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MgtTxFailed           | Number of HW Tx errors and excessive retries for management frame.                                                                                                                                                                             |
| [2.4G radio only] Rx  | Number of data MPDUs received for each modulation coding scheme (MCS).                                                                                                                                                                         |
| TxOffer               | Number of Ethernet frames at the entrance of the UMAC.                                                                                                                                                                                         |
| TxOfferBytes          | Number of bytes in the Ethernet frames at the entrance of the UMAC.                                                                                                                                                                            |
| STATS_ACTIVITY_FACTOR | Percentage of time the radio was unable to transmit or receive Wi-Fi packets to/from associated clients due to energy detection (ED) on the channel or clear channel assessment (CCA) in accordance to polling rate and interval, respectively |
| STATS_RETRANS_METRIC  | Percentage of packets that had to be re-transmitted. Multiple retransmissions of the same packet count as one. Calculated in accordance to the polling rate and interval,                                                                      |
| RxOffer               | Number of data MPDUs received from the radio.                                                                                                                                                                                                  |
| RxOfferBytes          | Number of bytes in the data MPDUs received from the radio.                                                                                                                                                                                     |
| RxFailed              | Number of data MPDUs missed by sequence number gap.                                                                                                                                                                                            |

|         |                                                                                                                                                             |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MinRSSI | The Minimum Received Signal Strength Indicator; RSSI parameter is the minimum energy observed at the antenna receiver for past transmissions (100 packets). |
| MaxRSSI | The Maximum Received Signal Strength Indicator; RSSI parameter is the energy observed at the antenna receiver for past transmissions (100 packets).         |

- The number of hardware Tx errors and excessive retries for management frames are displayed in the output of the apstats -a -R command at the radio level.
- For 2.4 GHz radios, received modulation coding scheme (MCS) packets need to be calculated; only 802.11n supports MCS. The value for rs\_rate in the ath\_rx\_status structure is in the range of 0x80-0x8f means MCS0-MCS15. But the rs\_rate is in the range of 0-255 because other rates, besides MCS, exist. A new variable, ast\_rx\_rate\_packets [256 bytes], is implemented to record packets received under different rs\_rate. The output of the athstats -i wifiX command displays the MCS Rx statistics.
- The TxOffer and TxOfferBytes values are displayed in the output of the apstats -a -R command at the radio level.
- The STATS\_ACTIVITY\_FACTOR value is displayed in the output of the iwpriv athx bss\_chan\_info 1 command.
- The STATS\_RETRANS\_METRIC value is displayed in the output of the athstats -i wifiX command.
- The “Rx Data Packets” value that corresponds to the data MPDUs received from the radio frames is displayed in the output of the apstats -a -R command at the radio level.
- The “Rx Data Bytes” value per radio that corresponds to the Rx offer bytes is displayed in the output of the apstats -a -R command at the radio level.
- The “Rx errors” value that corresponds to the number of data MPDUs that are failed to be received is displayed in the output of the apstats -a -R command at the radio level.
- The “Rx RSSI” value that corresponds to the minimum and maximum RSSI is displayed in the output of the apstats -a -R command at the radio level.

## 22.15 Integration of Wi-Fi statistics with vendors for extended statistics

An infrastructure has been implemented for NSS offload to update the statistics in the Wi-Fi host. The Wi-Fi driver provides a infrastructure to provide extended stats according to MCS/BW/NS. Customers or vendors (for example, Plume) need to store the values in their desired way or use for any additional algorithm.

Enter the `iwpriv wifiX enable_ol_stats 1` command to enable the offload-mode statistics. Subsequently, enter the `iwpriv wifiX enable_statsv2 <Value>` command to selectively enable different statistics or use 0xF to enable all statistics. The following values are possible:

- Bit0: WIFI\_RX\_STAT\_A\_REGULAR\_PEER - Default non zero value, Rx Extended stats for regular peer updated

- Bit1: WIFI\_RX\_STATS\_STATUS\_NOT\_OK - Rx extended statistics for Errored/FCS packets are updated
- Bit2: WIFI\_RX\_STATS\_INTRABSS - Rx statistics for intra-BSS frames are updated
- Bit3: WIFI\_TX\_STATS\_TIMESTAMP - Tx Sojourn statistics are updated

To enable all statistics, enter the following command:

```
iwpriv wifi0 enable_statsv2 0xF
```

To obtain the statistical details, enter the `apstats -a -R` command.

In the following table, the green-shaded cells indicate that the vendors can continue to use/test them in any release as today, and Qualcomm implementation remains the same. The yellow-shaded cells indicate the Qualcomm-supported infrastructure.

There are two sets of statistics—generic statistics that the Wi-Fi driver provides that are denoted by the green-shaded cells and extended statistics that are denoted by the yellow-colored cells. After the customer expands the statistics for their diagnostic and computation purposes, these extended statistics can be sent to any destination, such as the cloud. Because the per-PPDU information is retrieved by the Wi-Fi driver, and if this information is needed to be stored per client, per rate, and per MCS, the data structure becomes multidimensional. Storing such information per client needs more size. As a result, the `apstats` tool displays only some (generic) statistics. The customer can use their infrastructure to store extended stats per client, per rate (MCS), or per bandwidth.

**Table 22-2 Offload statistics parameters**

| Parameter              | Command                                                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------|--------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enable_ol_stats        | <code>iwpriv wifiX enable_ol_stats</code>              | Enter the <code>iwpriv wifiX enable_ol_stats 1</code> command to enable the collection of offload-mode statistics. Only after the <code>iwpriv wifi0 enable_ol_stats 1</code> command is issued to enable station statistics, the output of the <code>apstats -s -m</code> command displays information pertaining to AP statistical details.                                                                                                                                                                                                                                                                                                                                                                                               |
| enable_statsv2 <Value> | <code>iwpriv wifiX enable_statsv2 &lt;Value&gt;</code> | <p>After enabling the offload-mode statistics, enter this command to selectively enable different statistics or use 0xF to enable all statistics. The following values are possible:</p> <ul style="list-style-type: none"> <li>■ Bit0: WIFI_RX_STATS_REGULAR_PEER - Default non zero value, Rx Extended stats for regular peer updated</li> <li>■ Bit1: WIFI_RX_STATS_STATUS_NOT_OK - Rx extended statistics for Errored/FCS packets are updated</li> <li>■ Bit2: WIFI_RX_STATS_INTRABSS - Rx statistics for intra-BSS frames are updated</li> <li>■ Bit3: WIFI_TX_STATS_TIMESTAMP - Tx Sojourn statistics are updated</li> </ul> <p>To enable all statistics, enter the following command:</p> <pre>iwpriv wifi0 enable_statsv2 0xF</pre> |

| Report    | Mode                                      | Parameter     | Current vendor implementation                                                                                                     | Qualcomm changes                                                                                                                                                                       | Vendor changes                                                              | IOCTL                                                                   |
|-----------|-------------------------------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|-------------------------------------------------------------------------|
|           | radio information                         | band          |                                                                                                                                   |                                                                                                                                                                                        |                                                                             |                                                                         |
| NEIGHBOUR | on-channel<br>off channel<br>full-channel | bssid         | standard IOCTL<br>SIOCSIWSCAN<br>SIOCWIWSCAN                                                                                      | No Change                                                                                                                                                                              | No Change                                                                   | SIOCSIWSCAN<br>SIOCWIWSCAN                                              |
|           |                                           | ssid          | Vendors can continue to use the standard IOCTL                                                                                    |                                                                                                                                                                                        |                                                                             |                                                                         |
|           |                                           | rssi          |                                                                                                                                   |                                                                                                                                                                                        |                                                                             |                                                                         |
|           |                                           | tsf           |                                                                                                                                   |                                                                                                                                                                                        |                                                                             |                                                                         |
|           |                                           | channel width |                                                                                                                                   |                                                                                                                                                                                        |                                                                             |                                                                         |
|           |                                           | channel       |                                                                                                                                   |                                                                                                                                                                                        |                                                                             |                                                                         |
| SURVEY    | on-channel<br>off channel<br>full-channel | active        | Vendor implemented these by reading cycle counters from MAC_PCU registers for every scan event or when user issues ioctl.<br>+C54 | Qualcomm to provide per-packet Rx Packet header TLVs for calculating average rx_duration.                                                                                              | For Rx duration, Vendor to extract information from extended Rx statistics. | Vendor extended based on CC counters and time receiving Vendor packets! |
|           |                                           | busy          | For busy_self, rx_duration is estimated by extracting rate information from frame descriptor and packet length.                   | For Rx duration calculation, refer to extended Rx statistics.<br>Optionally, Vendor can use WLAN f/w event which provides these information taking into consideration of pkt duration. |                                                                             |                                                                         |
|           |                                           | busy_tx       |                                                                                                                                   |                                                                                                                                                                                        |                                                                             |                                                                         |
|           |                                           | busy_rx       |                                                                                                                                   |                                                                                                                                                                                        |                                                                             |                                                                         |
|           |                                           | busy_self     |                                                                                                                                   |                                                                                                                                                                                        |                                                                             |                                                                         |
|           |                                           | busy_ext      |                                                                                                                                   |                                                                                                                                                                                        |                                                                             |                                                                         |
| Clients   | Client information                        | mac_address   | IEEE80211_IOCTL_STA_INFO<br>Vendor can continue using standard IOCTL                                                              | No Change                                                                                                                                                                              | No Change                                                                   | IEEE80211_IOCTL_STA_INFO                                                |
|           |                                           | ssid          |                                                                                                                                   |                                                                                                                                                                                        |                                                                             |                                                                         |
|           |                                           | band          |                                                                                                                                   |                                                                                                                                                                                        |                                                                             |                                                                         |

| Report | Mode                 | Parameter  | Current vendor implementation                                                                                                                                                                                             | Qualcomm changes                                                                                 | Vendor changes | IOCTL         |
|--------|----------------------|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|----------------|---------------|
|        | Average driver stats | rx_bytes   | SIOCG80211STATS ,<br>Enable OL stats need to be enabled.<br><br>Vendor can continue using the standard IOCTL.<br><br>To support extended statistics per client per rate the internal implementation is expected to change | Qualcomm will update these statistics in same Host driver structures, in NSS offload model also. | No Change      | SIOCG80211STS |
|        |                      | tx_bytes   |                                                                                                                                                                                                                           |                                                                                                  |                |               |
|        |                      | rx_msdu    |                                                                                                                                                                                                                           |                                                                                                  |                |               |
|        |                      | tx_msdu    |                                                                                                                                                                                                                           |                                                                                                  |                |               |
|        |                      | rx_retries |                                                                                                                                                                                                                           |                                                                                                  |                |               |
|        |                      | tx_retries |                                                                                                                                                                                                                           |                                                                                                  |                |               |
|        |                      | rx_errors  |                                                                                                                                                                                                                           |                                                                                                  |                |               |
|        |                      | tx_errors  |                                                                                                                                                                                                                           |                                                                                                  |                |               |
|        |                      | rx_rate    |                                                                                                                                                                                                                           |                                                                                                  |                |               |
|        |                      | tx_rate    |                                                                                                                                                                                                                           |                                                                                                  |                |               |
|        |                      | rssi       |                                                                                                                                                                                                                           |                                                                                                  |                |               |

| Report | Mode                                 | Parameter | Current vendor implementation                                                                                                                                                                                              | Qualcomm changes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Vendor changes                                                                                                                                                                                                                                                                                       | IOCTL                           |
|--------|--------------------------------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
|        | Extended RX stats per MCS MCS/NSS/BW | mcs       | <p>These are extracted from per-packet msdu/mpdu/ppdu status TLVs in the Rx Packet Header in ol_rx_indication handler.</p> <p>peer_id is obtained from HTT header, and node lookup is done using peer-&gt;mac_address.</p> | <p>Qualcomm will provide per-packet rx_descriptors in same format, along with meta header containing additional information from which Vendor can derive the necessary information</p> <p>Example can be found at osif_nss_ni_stats_update</p> <p>The information is also sent for intrabss and Rx erro packet</p> <p>(1) Qualcomm provides complete Rx descriptor of all MSDU as part of packet as it exist today.</p> <p>(2) In addition to above NSS F/w calculates the below fields per PPDU received and update the information as a meta header only for head MSDU of the 1st MPDU of the PPDU</p> <p>3. Qualcomm offers sample functions, where Vendor can add its own code.</p> | <p>Vendor can add their function : <b>osif_nss_ni_stats_update</b></p> <p>This API will be called per-MSDU and not per-MPDU</p> <p>With Rx descriptor of 1stMSDU of 1st MPDU (of an AMPDU) and the metaheader information Vendor must be able to retrieve all information to fill in these stats</p> | PS_UAPI_IOCTL_CMD_PEER_RX_STATS |

| Report | Mode | Parameter | Current vendor implementation | Qualcomm changes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Vendor changes | IOCTL |
|--------|------|-----------|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|-------|
|        |      | nss       |                               | <pre> /*  * nss_wifi_append_ metaheader  * Append metaheader after pbuf-&gt;data for stats_v2.  */ struct nss_wifi_append_ statsv2_metahdr {     uint32_t rxstatsmagic; /* Magic to be verified on host */     uint32_t seq_number; /* sequence number of packets sent from nss */     uint16_t peer_id; /* peer_id of peer */     uint16_t num_msdu; /* msdus in ppdu */     uint16_t num_retries; /* retries in ppdu */     uint16_t num_mpdu; /* mpdus in ppdu */     uint32_t num_bytes; /* bytes in ppdu */ };  Vendor checks for the valid field in meta header to identify other fields are valid and this packet is the head MSDU of the 1st MPDU of the PPDU.  With this information, Vendor must be able to populate all the table.  Vendor needs to inspect only the Head MSDU of 1st MPDU instead of inspecting for all the MSDU </pre> |                |       |

| Report | Mode | Parameter      | Current vendor implementation | Qualcomm changes | Vendor changes | IOCTL |
|--------|------|----------------|-------------------------------|------------------|----------------|-------|
|        |      | bw             |                               |                  |                |       |
|        |      | rx_retries     |                               |                  |                |       |
|        |      | mpdu           |                               |                  |                |       |
|        |      | ppdu           |                               |                  |                |       |
|        |      | retries        |                               |                  |                |       |
|        |      | errors         |                               |                  |                |       |
|        |      | rssi           |                               |                  |                |       |
|        |      | per_chain_rssi |                               |                  |                |       |

| Report | Mode                                    | Parameter | Current vendor implementation                                                                         | Qualcomm changes                                                                                                       | Vendor changes | IOCTL                           |
|--------|-----------------------------------------|-----------|-------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|----------------|---------------------------------|
|        | Extended TX stats per MCS<br>MCS/NSS/BW | mcs       | These are extracted from per-PPDU stats sent from Firmware ( <code>ol_ath_enh_stats_handler</code> ). | Same event handler interface ( <code>ol_ath_enh_stats_handler</code> ) will be implemented for NSS offload model also. | No change      | PS_UAPI_IOCTL_CMD_PEER_TX_STATS |
|        |                                         | nss       |                                                                                                       |                                                                                                                        |                |                                 |
|        |                                         | bw        |                                                                                                       |                                                                                                                        |                |                                 |
|        |                                         | bytes     |                                                                                                       |                                                                                                                        |                |                                 |
|        |                                         | mpdus     |                                                                                                       |                                                                                                                        |                |                                 |

| Report | Mode                                         | Parameter    | Current vendor implementation | Qualcomm changes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Vendor changes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | IOCTL |
|--------|----------------------------------------------|--------------|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
|        | Extended TX stats (sojourn) per TID per Peer | ewma_time_ms |                               | <p>NSS will send these statistics to Host in periodic events (period &lt; 1 second).</p> <p>NSS will do the averaging for soujourn_msec.</p> <p>Vendor can enable+E56 and disable with the command iwpriv wifi0 enable_statsv2 0xF</p> <p>The EWMA weight and factor used</p> <pre>WIFI_TX_SOJOURN_EWMA_FACTOR 10 WIFI_TX_SOJOURN_EWMA_WEIGHT 3</pre> <p>Average calculation</p> <pre>local_avg = local_avg ? (((local_avg &lt;&lt; WIFI_TX_SOJOURN_EWMA_WEIGHT) - local_avg) + (delta_ms &lt;&lt; WIFI_TX_SOJOURN_EWMA_FACTOR)) &gt;&gt; WIFI_TX_SOJOURN_EWMA_WEIGHT : (delta_ms &lt;&lt; WIFI_TX_SOJOURN_EWMA_FACTOR);</pre> <p>If required, Qualcomm can provide a separate patch to configure the factor and weight</p> | <p>NSS offload updates the host peer structure with required information periodically. Vendor can read the host peer structure for required information</p> <pre>struct ol_txrx_peer_t {     ...     struct ol_peer_tstamp_stats tstampstat; };  struct ol_peer_tstamp_stats {     uint64_t sum[OL_TXRX_NUM_EXT_TIDS]; /* micro second */     uint64_t nummsdu[OL_TXRX_NUM_EXT_TIDS];     uint64_t avg[OL_TXRX_NUM_EXT_TIDS]; /*millisecond */ };  sum is provided in micro seconds avg is provided in millisecond For accurate timestamp information, Vendor needs to disable NSS Auto Power scaling and Keep the NSS at Max Freq 800Mhz with below commands.</pre> |       |
|        |                                              | sum_time_ms  |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |       |

| Report   | Mode              | Parameter    | Current vendor implementation                                                                             | Qualcomm changes | Vendor changes | IOCTL                                                                |
|----------|-------------------|--------------|-----------------------------------------------------------------------------------------------------------|------------------|----------------|----------------------------------------------------------------------|
|          |                   | num_msdu     |                                                                                                           |                  |                |                                                                      |
| CAPACITY | radio information | band         |                                                                                                           |                  | N/A            | Vendor extended based on sampling queues and checking the busy flag! |
|          | Phy utilization   | busy_tx      | Vendor extended based on sampling queues and checking the busy flag.<br>Vendor can continue with the same |                  |                |                                                                      |
|          |                   | active       |                                                                                                           |                  |                |                                                                      |
|          |                   | tx_bytes     |                                                                                                           |                  |                |                                                                      |
|          | queue busyness    | sample count |                                                                                                           |                  |                |                                                                      |

| Report | Mode                | Parameter  | Current vendor implementation                                                                                                                                                                                                                                                            | Qualcomm changes | Vendor changes | IOCTL                                                                                                                                                                                                                                         |
|--------|---------------------|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        |                     | VO_count   |                                                                                                                                                                                                                                                                                          |                  |                |                                                                                                                                                                                                                                               |
|        |                     | VI_count   |                                                                                                                                                                                                                                                                                          |                  |                |                                                                                                                                                                                                                                               |
|        |                     | BE_count   |                                                                                                                                                                                                                                                                                          |                  |                |                                                                                                                                                                                                                                               |
|        |                     | BK_count   |                                                                                                                                                                                                                                                                                          |                  |                |                                                                                                                                                                                                                                               |
|        |                     | Cab_count  |                                                                                                                                                                                                                                                                                          |                  |                |                                                                                                                                                                                                                                               |
|        |                     | BCN_count  |                                                                                                                                                                                                                                                                                          |                  |                |                                                                                                                                                                                                                                               |
| RADIO  | Radio configuration | band       | SIOCGIWNNAME<br>SIOCGIFHWADDR<br>SIOCGIWAP<br>SIOCGIWESSIONID<br>SIOCGIWTXPOW<br>SIOCGIWTXPOW<br>IEEE80211_IOCTL_<br>GETMODE<br>ATH_IOCTL_<br>GETCOUNTRY<br>IEEE80211_IOCTL_<br>GETCHAN<br>INFO<br>ATH_PARAM_<br>RXCHAINMASK<br>... also set+D76<br>Vendor can continue with<br>the same |                  |                | SIOCGIWNNAME<br>SIOCGIFHWADDR<br>SIOCGIWAP<br>SIOCGIWESSIONID<br>SIOCGIWTXPOW<br>SIOCGIWTXPOW<br>IEEE80211_IOCTL_<br>GETMODE<br>ATH_IOCTL_<br>GETCOUNTRY<br>IEEE80211_IOCTL_<br>GETCHAN<br>INFO<br>ATH_PARAM_<br>RXCHAINMASK<br>... also set! |
|        |                     | phy_name   |                                                                                                                                                                                                                                                                                          |                  |                |                                                                                                                                                                                                                                               |
|        |                     | if_name    |                                                                                                                                                                                                                                                                                          |                  |                |                                                                                                                                                                                                                                               |
|        |                     | chan       |                                                                                                                                                                                                                                                                                          |                  |                |                                                                                                                                                                                                                                               |
|        |                     | chan_mode  |                                                                                                                                                                                                                                                                                          |                  |                |                                                                                                                                                                                                                                               |
|        |                     | chan_width |                                                                                                                                                                                                                                                                                          |                  |                |                                                                                                                                                                                                                                               |

| Report | Mode                     | Parameter    | Current vendor implementation                                                                                             | Qualcomm changes | Vendor changes | IOCTL                           |
|--------|--------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------|------------------|----------------|---------------------------------|
|        |                          | tx_pwr       |                                                                                                                           |                  |                |                                 |
|        |                          | cntry_code   |                                                                                                                           |                  |                |                                 |
|        |                          | protocol     |                                                                                                                           |                  |                |                                 |
|        |                          | admin_status |                                                                                                                           |                  |                |                                 |
|        | Radio stats              | rx_bytes     | SIOCG80211STATS<br>SIOCGATHSTATS<br><br>Vendor can continue use the same.<br>Qualcomm enable ol stats need to be enabled. |                  |                | SIOCG80211STS<br>SIOCGATHSTA TS |
|        |                          | tx_bytes     |                                                                                                                           |                  |                |                                 |
|        |                          | rx_frames    |                                                                                                                           |                  |                |                                 |
|        |                          | tx_frames    |                                                                                                                           |                  |                |                                 |
|        |                          | rx_mc_frame  |                                                                                                                           |                  |                |                                 |
|        |                          | tx_mc_frame  |                                                                                                                           |                  |                |                                 |
|        |                          | noise_floor  |                                                                                                                           |                  |                |                                 |
|        |                          | rx_beacon    |                                                                                                                           |                  |                |                                 |
| ESSID  | ESSID config/information | band         | hostapd<br>Vendor can continue using the same                                                                             |                  |                | hostapd                         |
|        |                          | ssid         |                                                                                                                           |                  |                |                                 |
|        |                          | bssid        |                                                                                                                           |                  |                |                                 |
|        |                          | if_name      |                                                                                                                           |                  |                |                                 |
|        |                          | security     |                                                                                                                           |                  |                |                                 |
|        |                          | passphrase   |                                                                                                                           |                  |                |                                 |
|        |                          | hide         |                                                                                                                           |                  |                |                                 |
|        |                          | isolate      |                                                                                                                           |                  |                |                                 |
|        |                          | radius       |                                                                                                                           |                  |                |                                 |

| Report | Mode        | Parameter  | Current vendor implementation                                                                                                                                               | Qualcomm changes | Vendor changes | IOCTL                                                                    |
|--------|-------------|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|----------------|--------------------------------------------------------------------------|
|        | ESSID stats | rx_bytes   | fopen<br>(PROC_NET_WIRELESS,<br>"r")<br><br>SIOCGIWSTATS<br>SIOCG80211STATS<br><br>Qualcomm enable ol stats<br>need to be enabled.<br>Vendor can continue using<br>the same |                  |                | fopen<br>(PROC_NET_WIRELESS, "r")<br><br>SIOCGIWSTATS<br>SIOCG80211STATS |
|        |             | tx_bytes   |                                                                                                                                                                             |                  |                |                                                                          |
|        |             | rx_frames  |                                                                                                                                                                             |                  |                |                                                                          |
|        |             | tx_frames  |                                                                                                                                                                             |                  |                |                                                                          |
|        |             | rx_retries |                                                                                                                                                                             |                  |                |                                                                          |
|        |             | tx_retries |                                                                                                                                                                             |                  |                |                                                                          |
|        |             | rx_errors  |                                                                                                                                                                             |                  |                |                                                                          |
|        |             | tx_errors  |                                                                                                                                                                             |                  |                |                                                                          |
|        |             | clients    |                                                                                                                                                                             |                  |                |                                                                          |

| Report | Mode                 | Parameter | Plume Changes | IOCTL |
|--------|----------------------|-----------|---------------|-------|
|        | radio<br>information | band      |               |       |

|               |                                           |                                                                                                                                           |                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------|-------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NEIGH<br>BOUR | on-channel<br>off channel<br>full-channel | bssid                                                                                                                                     | iwlist athx scan                                                                                                                                                                                                | Scan completed :                                                                                                                                                                                                                                                                                                                                                                            |
|               |                                           | ssid                                                                                                                                      |                                                                                                                                                                                                                 | Cell 01 - Address: 8C:FD:F0:02:13:63<br>ESSID:"Cascade-5G"                                                                                                                                                                                                                                                                                                                                  |
|               |                                           | rssi                                                                                                                                      |                                                                                                                                                                                                                 | Mode:Master                                                                                                                                                                                                                                                                                                                                                                                 |
|               |                                           | tsf                                                                                                                                       |                                                                                                                                                                                                                 | Frequency:5.18 GHz (Channel 36)                                                                                                                                                                                                                                                                                                                                                             |
|               |                                           | channel width                                                                                                                             |                                                                                                                                                                                                                 | Quality=94/94 Signal level=-51 dBm Noise level=-95 dBm<br>Encryption key:off<br>Bit Rates:6 Mb/s; 9 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s<br>36 Mb/s; 48 Mb/s; 54 Mb/s<br>Extra:bcn_int=100<br>Extra:wme_                                                                                                                                                                                         |
| Clients       | Client<br>information                     | channel                                                                                                                                   |                                                                                                                                                                                                                 | ie=dd180050f2020101800003a4000027a4000042435e0062322f00<br>Extra:phy_mode=IEEE80211_MODE_11AC_VHT80<br>Extra:ath_ie=dd0900037f01010000ff7f<br>Extra:dtim_period=1                                                                                                                                                                                                                           |
|               |                                           | mac_address                                                                                                                               | wlanconfig athx list<br>sta                                                                                                                                                                                     | IEADDR AID CHAN TXRATE RXRATE RSSI MINRSSI MAXRSSI IDLE TXSEQ RXSEQ<br>CAPS ACAPS ERP STATE MAXRATE(DOT11) HTCAPS ASSOCTIME IEs MODE<br>PSMODE<br>8c:fd:f0:02:0b:47 1 36 0M 0M -2 0 71 11 0 65535 Es 0 b<br>0 AWPSM 00:00:14 WME IEEE80211_MODE_11AC_VHT160 0                                                                                                                               |
|               |                                           | ssid                                                                                                                                      |                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                             |
|               |                                           | band                                                                                                                                      |                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                             |
|               | Average driver<br>stats                   | rx_bytes<br>rx_bytes<br>tx_bytes<br>rx_msdu<br>tx_msdu<br>rx_retries<br>tx_retries<br>rx_errors<br>tx_errors<br>rx_rate<br>tx_rate<br>rss | Configure DUT in mode. Connect n numbers of stations.<br>Disable ECM : /etc/init.d/qca-nss-ecm stop<br>Enable stats: iwpriv wifiX enable_ol_stats 1<br>iwpriv wifiX enable_statsv2 0xF<br>Check : apstats -a -R | Node Level Stats: 8c:fd:f0:02:14:b6 (under VAP ath0)<br>Tx Data Packets = 775<br>Tx Data Bytes = 58320<br>Rx Data Packets = 1141<br>Rx Data Bytes = 95578<br>Rx retry count = 21<br>Rx mpdu count = 913<br>Rx ppdu count = 913<br>Rx errors = 0<br>Average Tx Rate (kbps) = 1733000<br>Average Rx Rate (kbps) = 1733000<br>Last tx rate = 1733000<br>Last rx rate = 1733000<br>Rx RSSI = 61 |

|         |                                      |                                                                                                                                                                                                                     |                                                                                               |                                                      |
|---------|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|------------------------------------------------------|
|         |                                      |                                                                                                                                                                                                                     |                                                                                               | Node Level Stats: 8c:fd:f0:02:0b:47 (under VAP ath0) |
|         |                                      |                                                                                                                                                                                                                     | Rx mpdu count                                                                                 | = 10                                                 |
|         |                                      |                                                                                                                                                                                                                     | Rx ppdu count                                                                                 | = 10                                                 |
|         |                                      |                                                                                                                                                                                                                     | Rx retry count                                                                                | = 0                                                  |
|         |                                      |                                                                                                                                                                                                                     | Rx errors                                                                                     | = 0                                                  |
|         |                                      |                                                                                                                                                                                                                     | Rx RSSI                                                                                       | = 67                                                 |
|         |                                      |                                                                                                                                                                                                                     | Ack RSSI chain 1                                                                              | = 65                                                 |
|         |                                      |                                                                                                                                                                                                                     | Ack RSSI chain 2                                                                              | = 61                                                 |
|         |                                      |                                                                                                                                                                                                                     | Ack RSSI chain 3                                                                              | = 45                                                 |
|         |                                      |                                                                                                                                                                                                                     | Ack RSSI chain 4                                                                              | = 61                                                 |
|         |                                      |                                                                                                                                                                                                                     | Band Width                                                                                    | = 80                                                 |
| Clients | Extended RX stats per MCS MCS/NSS/BW | Configure DUT in mode. Connect n numbers of stations.<br>Disable ECM : /etc/init.d/qca-nss-ecm stop<br>Enable stats: iwpriv wifiX enable.ol_stats 1<br><br>iwpriv wifiX enable.statsv2 0xF<br>Check : apstats -a -R | mcs<br>nss<br>bw<br>rx_retries<br>mpdu<br>ppdu<br>retries<br>errors<br>rssI<br>per_chain_rssi | 2017-11-01 18:46:38 PST<br>wei.wu@eacsz.com.cn       |

|         |                                                                                     |  |                                                                                                                                                                       |                                                                                                                                                                                                                                                                   |
|---------|-------------------------------------------------------------------------------------|--|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|         |                                                                                     |  | Customer can get stats from enhanced stats function as packet is sent to host. These are extracted from per-PPDU stats sent from Firmware (ol_ath_enh_stats_handler). | PS_UAPI_IOCTL_CMD_PEER_TX_STATS                                                                                                                                                                                                                                   |
| Clients | <p>Extended TX stats per MCS<br/>MCS/NSS/BW</p> <p>mcs<br/>nss<br/>bw<br/>bytes</p> |  |                                                                                                                                                                       | <pre> iwppriv wifi0 fc_peer_stats 8c:fd:f0:02:0b:47 Found peer_id=1 Peer Stats Message - success 0 ----- Assoc ID = 1 ----- ----- per-Peer/TIDQ Stats ----- ----- peer_id 1, tid 0 -----+-----  Pkts stay time  277208 Msdu stayed   416 EWMA time diff  0 </pre> |

|              |                      |              |                                                                                                                            |                                                                           |
|--------------|----------------------|--------------|----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| CAPAC<br>ITY | radio<br>information | band         | Plume extended<br>based on<br>sampling queues<br>and checking<br>the busy flag!<br><br>Plume can continue<br>with the same | Plume extended based on<br>sampling queues and checking<br>the busy flag! |
|              | Phy utilization      | busy_tx      |                                                                                                                            |                                                                           |
|              |                      | active       |                                                                                                                            |                                                                           |
|              |                      | tx_bytes     |                                                                                                                            |                                                                           |
|              |                      | sample count |                                                                                                                            |                                                                           |
|              |                      | VO_count     |                                                                                                                            |                                                                           |
|              |                      | VI_count     |                                                                                                                            |                                                                           |
|              |                      | BE_count     |                                                                                                                            |                                                                           |
|              |                      | BK_count     |                                                                                                                            |                                                                           |
|              |                      | Cab_count    |                                                                                                                            |                                                                           |
|              |                      | BCN_count    |                                                                                                                            |                                                                           |

|                     |             |              |                                                       |                                                                     |
|---------------------|-------------|--------------|-------------------------------------------------------|---------------------------------------------------------------------|
|                     |             | band         | iwconfig                                              |                                                                     |
|                     |             | phy_name     | iwpriv athx get_mode                                  | ath0 get_mode:11ACVHT80<br>wifi0 getCountry:US                      |
|                     |             | if_name      | iwpriv wifi0 getCountry                               | Channel 44 : 5220 Mhz 11na C CU V VU V80- 42 V160- 50               |
|                     |             | chan         | wlanconfig ath0 list chan                             | wifi0 get_rxchainmask:15                                            |
|                     |             | chan_mode    |                                                       |                                                                     |
|                     |             | chan_width   |                                                       |                                                                     |
|                     |             | tx_pwr       |                                                       |                                                                     |
|                     |             | admin_status |                                                       |                                                                     |
|                     |             | cntry_code   |                                                       |                                                                     |
| Radio configuration | Radio stats | rx_bytes     | Configure DUT in mode. Connect n numbers of stations. | AP Level Stats:<br><br>WLAN Stats:                                  |
|                     |             | tx_bytes     |                                                       | Tx Data Packets = 15041<br>Tx Data Bytes = 1586400                  |
|                     |             | rx_frames    | Disable ECM : /etc/init.d/qca-nss-ecm stop            | Rx Data Packets = 1921<br>Rx Data Bytes = 153520                    |
|                     |             | tx_frames    | Enable stats: iwpriv wifiX enable_ol_stats 1          | Tx Multi/Broadcast Data Packets = 13117<br>Rx Mgmt Frames = 7583234 |
|                     |             | rx_mc_frame  |                                                       |                                                                     |
|                     |             | tx_mc_frame  |                                                       |                                                                     |
|                     |             | noise_floor  |                                                       | athstats:<br>noise floor : -104<br>noise floor (sec 80): 0          |
|                     |             |              | iwpriv wifiX enable_statsv2 0xF                       |                                                                     |
|                     |             | rx_beacon    | Check : apstats -a -R<br>For noise_floor:<br>athstats |                                                                     |

|       |                          |            |                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------|--------------------------|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|       | ESSID config/information | band       | iwconfig<br>iwpriv ath0 get_hide_ssid                                                                                                                                                                                                                       | ath0 IEEE 802.11a ESSID:"ssid_new"<br>Mode:Master Frequency:5.18 GHz Access Point: 8C:FD:F0:02:0B:56<br>Bit Rate:54 Mb/s Tx-Power:26 dBm<br>RTS thr:off Fragment thr:off<br>Encryption key:1234-5678-9012-3456-7890-1234-56 Security mode:restricted<br>Power Management:off<br>Link Quality=94/94 Signal level=-97 dBm Noise level=-95 dBm<br>Rx invalid nwid:38 Rx invalid crypt:0 Rx invalid frag:0<br>Tx excessive retries:0 Invalid misc:0 Missed beacon:0 |
|       |                          | ssid       |                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|       |                          | bssid      |                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|       |                          | if_name    |                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|       |                          | security   |                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|       |                          | passphrase |                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|       |                          | hide       |                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| ESSID | ESSID stats              | rx_bytes   | Configure DUT in mode. Connect n numbers of stations.<br>Disable ECM : /etc/init.d/qca-nss-ecm stop<br>Enable stats: iwpriv wifiX enable_ol_stats 1<br>iwpriv wifiX enable_statsv2 0xF<br>Check : apstats -a -R<br>For clients:<br>wlanconfig athx list sta | VAP Level Stats: ath0 (under radio wifi0)<br>Tx Data Packets = 15041<br>Tx Data Bytes = 1586400<br>Rx Data Packets = 1921<br>Rx Data Bytes = 153520<br>Retries = 3293<br>Rx Dropped = 0<br>Rx errors = 0<br>Tx failures = 4                                                                                                                                                                                                                                     |

## 22.16 Display of decoded watchdog timer signature for QCA9980

The capability to display decoded signature of watchdog timer is implemented. This functionality enables a user to identify which state machine caused the issue to trigger Watchdog timer. Before the implementation of this functionality to print the decoded signature of watchdog timer, the watchdog timer signature was displayed when it was triggered. The following is a sample of such a log:

### Format:

```
WAL_DBGID_BB_WDOG_TRIGGERED (time, bb_wdog_status, bb_wdog_status_b, g_dbg_num_bb_timeout);
```

```
WAL_DBGID_BB_WDOG_TRIGGERED (bb_wdog_status_c, bb_wdog_status_d);
```

**Example:**

```
[ 1584.333899] [wifi0] FWLOG: [279277] WAL_DBGID_BB_WDOG_TRIGGERED (0x2edaa, 0x43000309, 0x0, 0x4)
[ 1584.342643] [wifi0] FWLOG: [279277] WAL_DBGID_BB_WDOG_TRIGGERED ( 0x0, 0x0 )
```

Starting with QCA\_Networking\_2017.SPF.5.0 CSU1, the decoded watchdog timer signature is displayed, in addition to the existing logs. The message ID used to print is WAL\_DBGID\_RdOrCrOtCtAgSrHgRdsSsAbFoFbFp and its expansion is as follows:

Rd = Radar SM

Or = Ofdm Rx SM

Cr = Cck Rx SM

Ot = Ofdm Tx SM

Ct = Cck Tx SM

Ag = AGC SM

Sr = SRCH

Hg = HANG Flag

Rds = Radar Scan SM

Ss = Spectral Scan

Ab = Arb Lock

Fo = FFT Out

Fb = FFT BW

Fp = FFT Pwr

Refer the table at the end of this section for the status of each state machine.

**Format**

WAL\_DBGID\_RdOrCrOtCtAgSrHgRdsSsAbFoFbFp ( status delimited by value 8 )

**Example**

```
[ 2388.706832] [wifi1] FWLOG: [17326] WAL_DBGID_RdOrCrOtCtAgSrHgRdsSsAbFoFbFp ( 0x858f818a, 0x81828183, 0x81828780, 0x8681 )
```

From the preceding log, the base band (BB) watchdog timer state machine status can be decoded as follows:

|        |                                                                      |
|--------|----------------------------------------------------------------------|
| Status | How to read status from log WAL_DBGID_RdOr-CrOtCtAgSrHgRdsSsAbFoFbFp |
|--------|----------------------------------------------------------------------|

1. Rd(Radar SM) = 5 (0x858f818a, 0x81828183, 0x81828780, 0x8681)

- |                           |                                              |
|---------------------------|----------------------------------------------|
| 2. Or(OFDM RXSM) = f      | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |
| 3. Cr(CCK RXSM) = 1       | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |
| 4. Ot(OFDM TXSM) = 10     | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |
| 5. Ct(CCK TXSM) = 1       | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |
| 6. Ag(AGC) = 2            | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |
| 7. Sr(SRCH) = 1           | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |
| 8. Hg(HANG Flag) = 3      | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |
| 9. Rds(Radar Scan) = 1    | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |
| 10. Sc(Spectral Scan) = 2 | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |
| 11. Ab(ARB LOCK) = 7      | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |
| 12. Fo(FFT Out) = 0       | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |
| 13. Fb(FFT BW) = 6        | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |
| 14. Fp(FFT PWR) = 1       | (0x858f818a, 0x81828183, 0x81828780, 0x8681) |

### 22.16.1 Mapping of decoded watchdog timer debug log

The following table presents the mapping of Watchdog signature message fields and their respective states.

|   | State Machine | State          | Value |
|---|---------------|----------------|-------|
| 1 | wd_status     | IDLE           | 0     |
|   |               | ACTIVE TX/RX   | 1     |
|   |               | SEARCH MODE    | 2     |
|   |               | WAIT FOR CLEAR | 3     |
|   |               |                |       |

|  |  |                        |   |
|--|--|------------------------|---|
|  |  | IDLE                   | 0 |
|  |  | READY                  | 1 |
|  |  | WAIT A                 | 2 |
|  |  | WAIT B                 | 3 |
|  |  | WAIT C                 | 4 |
|  |  | RPT SHORT              | 5 |
|  |  | WAIT LONG              | 6 |
|  |  | RPT LONG               | 7 |
|  |  | RPT DONE               | 8 |
|  |  | WAIT WLAN              | 9 |
|  |  | WAIT SRCH              | A |
|  |  | UNDEF                  | B |
|  |  | UNDEF                  | C |
|  |  | UNDEF                  | D |
|  |  | UNDEF                  | E |
|  |  | TIME OUT               | F |
|  |  |                        |   |
|  |  | IDLE                   | 0 |
|  |  | TX                     | 1 |
|  |  | PREPEND CHAN INFO      | 2 |
|  |  | GET SIGNAL SYMBOL      | 3 |
|  |  | RX DATA                | 4 |
|  |  | RADAR                  | 5 |
|  |  | START DYN RX           | 6 |
|  |  | WAIT FOR RX CLEAR      | 7 |
|  |  | SPECTRAL SCAN          | 8 |
|  |  | CHANINFO DONE          | 9 |
|  |  | UNUSED                 | A |
|  |  | UNSUPPORTED HT RATE    | B |
|  |  | PILOT EVM              | C |
|  |  | HOLD                   | D |
|  |  | RX GREENFIELD          | E |
|  |  | STATIC20 MODE HT40 PKT | F |
|  |  |                        |   |

|   |                                  |          |   |
|---|----------------------------------|----------|---|
| 4 | CCK RXSM (wd_status_cck_rxsm)    | IDLE     | 0 |
|   |                                  | TX       | 1 |
|   |                                  | FR_DET   | 2 |
|   |                                  | PLSIG    | 3 |
|   |                                  | PLCP2    | 4 |
|   |                                  | PLCP3    | 5 |
|   |                                  | PSVC     | 6 |
|   |                                  | PLCP5    | 7 |
|   |                                  | RX       | 8 |
|   |                                  | ERROR    | 9 |
| 5 | OFDM TXSM (wd_status_ofdm_txsm)  | HOLD     | A |
|   |                                  | IDLE     | 0 |
|   |                                  | WAIT16   | 1 |
|   |                                  | L-SIG    | 2 |
|   |                                  | HT-SIG   | 3 |
|   |                                  | VHT-SIGB | 4 |
|   |                                  | DATA     | 5 |
|   |                                  | TAIL     | 6 |
|   |                                  | PAD      | 7 |
| 6 | CCK TXSM<br>(wd_status_cck_txsm) | WAIT     | 8 |
|   |                                  | IDLE     | 0 |
|   |                                  | WAIT     | 1 |
|   |                                  | SYNC     | 2 |
|   |                                  | HEADER   | 3 |
|   |                                  | DATA     | 4 |
|   |                                  | WAIT2    | 5 |

|   |                                       |                         |   |
|---|---------------------------------------|-------------------------|---|
| 7 | AGC<br>(wd_status_agc_sm)             | INIT                    | 0 |
|   |                                       | CALIBRATE               | 1 |
|   |                                       | IDLE                    | 2 |
|   |                                       | SEARCH                  | 3 |
|   |                                       | RECEIVE                 | 4 |
|   |                                       | WAIT                    | 5 |
|   |                                       | INIT_GAIN               | 6 |
|   |                                       |                         |   |
| 8 | SRCH<br>(wd_status_srch_sm)           | INIT                    | 0 |
|   |                                       | WEAK SIG DETECT         | 1 |
|   |                                       | STRONG SIG DETECT       | 2 |
|   |                                       | SEARCH GAIN SET         | 3 |
|   |                                       | SEARCH GAIN WAIT        | 4 |
|   |                                       | ANTENNA SWITCH          | 5 |
|   |                                       | ANTENNA SELECT          | 6 |
|   |                                       | FOUND GAIN CALC         | 7 |
|   |                                       | FOUND GAIN SET          | 8 |
|   |                                       | FOUND GAIN WAIT         | 9 |
|   |                                       | VOTE OFDM CCK           | A |
|   |                                       | DCOFF RESCALER WAIT CCK | B |
|   |                                       | SYNC FOUND WAIT CCK     | C |
|   |                                       | RESET ANT WAIT CCK      | D |
|   |                                       | WAIT SRCH FFT DONE      | E |
|   |                                       |                         |   |
| 9 | HANG FLAG<br>(wd_status_b_hang_flags) | NONE                    | 0 |
|   |                                       | RADAR_HANG              | 1 |
|   |                                       | SPECTRAL_HANG           | 2 |
|   |                                       | NONE                    | 3 |
|   |                                       | SRCH_FFT_HANG           | 4 |
|   |                                       |                         |   |

|    |                                              |                   |   |
|----|----------------------------------------------|-------------------|---|
| 10 | RADAR SCAN<br>(wd_status_b_radar_scan)       | IDLE              | 0 |
|    |                                              | READY             | 1 |
|    |                                              | WAIT_A            | 2 |
|    |                                              | WAIT_B            | 3 |
|    |                                              | WAIT_C            | 4 |
|    |                                              | RPT_SHORT         | 5 |
|    |                                              | WAIT_LONG         | 6 |
|    |                                              | RPT_LONG          | 7 |
|    |                                              | RPT_DONE          | 8 |
|    |                                              | WAIT_SRCH         | 9 |
|    |                                              | NONE              | A |
|    |                                              | NONE              | B |
|    |                                              | NONE              | C |
|    |                                              | NONE              | D |
|    |                                              | TIMEOUT           | E |
| 11 | SPECTRAL SCAN<br>(wd_status_b_spectral_scan) | IDLE              | 0 |
|    |                                              | READY             | 1 |
|    |                                              | WAIT_FFT          | 2 |
|    |                                              | RPT_FFT           | 3 |
|    |                                              | RPT_DONE          | 4 |
|    |                                              | ADCAP_RDY         | 5 |
|    |                                              | ADCAP_RTP         | 6 |
| 12 | ARB LOCK<br>(wd_status_b_arb_lock)           | NONE              | 0 |
|    |                                              | SPECTRAL_FFT_LOCK | 1 |
|    |                                              | RADAR_FFT_LOCK    | 2 |
| 13 | FFT OUT<br>(wd_status_b_search_fft_out)      | IDLE              | 0 |
|    |                                              | FFT_ON            | 1 |
|    |                                              | BW_ON             | 2 |
|    |                                              | RD_MEM            | 3 |

|    |                              |           |   |
|----|------------------------------|-----------|---|
| 14 | (wd_status_b_search_fft_bw)  | IDLE      | 0 |
|    |                              | READ      | 1 |
|    |                              | STATS     | 2 |
|    |                              | RPT_WAIT  | 3 |
|    |                              | RPT_HDR_A | 4 |
|    |                              | RPT_HDR_B | 5 |
|    |                              | RPT_HDR_C | 6 |
|    |                              | RPT_FFT   | 7 |
| 15 | (wd_status_b_search_fft_pwr) | MAX_SRCH  | 0 |
|    |                              | IP_POWR   | 1 |
|    |                              | OB_PWR    | 2 |
|    |                              | Avg_PWR   | 3 |

## 22.17 Smart logging for HALPHY modules

Support is implemented for a smart logging module for HALPHY failures, which reduces the problem or failure-reproduction effort of test teams and customers by collecting all the necessary logs for various possible error conditions. These failures can be calibration oriented such as DPD calibration failure, IBF failure, NF calibration failure or other failures such as Tx timeout, spectral scan, channel switch or reset issues. This failure scenario based logging will be responsible for the following:

- Identification of the trigger conditions.
- Collection of necessary debug information, for the failure conditions.
- Logging of vital information, only when there is a failure.
- This method of logging ensures that the necessary debug information is available at least on the console output for the identified error scenarios.

### 22.17.1 Host-FW interaction

Host and FW need to interact with each other for displaying trigger based smart logs on the console. The failure scenarios for which logs are captured are in DPD, IBF, noise floor failures, reset, and Tx timeouts.

**Table 22-3 Failure type and subtypes**

| Failure Type                 | Subtype                                                                                 | Identifier in Smart logs                                                     |
|------------------------------|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| Calibration                  | DPD calibration                                                                         | WAL_DBGID_DPD_STATS1<br>WAL_DBGID_DPD_STATS2<br>...<br>WAL_DBGID_DPD_STATS12 |
|                              | IBF                                                                                     | WAL_DBGID_IBF_STATS1<br>WAL_DBGID_IBF_STATS2<br>...<br>WAL_DBGID_IBF_STATS7  |
|                              | Noise Floor                                                                             | WAL_DBGID_NF_STATS1<br>WAL_DBGID_NF_STATS2<br>...<br>WAL_DBGID_NF_STATS7     |
| Tx timeout issues            |                                                                                         | WAL_DBGID_TX_TIMEOUT_STATS                                                   |
| Channel scan or reset issues |                                                                                         | WAL_DBGID_RST_STATS                                                          |
| Parameters                   | Parameters that the host sends to FW, whenever the FW returns a negative value to host. | TBD                                                                          |

## 22.17.2 Implementation details

Failure-based smart logging is carried on using existing DBGLOG\_RECORD\_LOG macro in HALPHY layer. In its input list, where there is a need to set the message ID, the values proposed in the table above shall be used. Multiple message IDs are created for each failure type because a single DBGLOG\_RECORD\_LOG macro call can log only five values. Because the smart logs is designed to create elaborate data to debug in case of a failure, this will be essential.

### 22.17.2.1 Codeswap of program and data

Because this new capability utilizes program memory more than the existing FW code, code swap modules are modified, freeing 150 bytes for each variant. Approximately 250 bytes of IRAM are left over after these changes for each variant.

## 22.17.3 Identification of calibration triggers—firmware

The errors mentioned in preceding table are detected by the firmware.

### 22.17.3.1 DPD calibration failure

In firmware, when a DPD failure occurs, 10 sessions of 5 retries have happened. In case this threshold is exceeded, DPD failure is signalled. In this scenario, multiple registers and state machine values need to be logged. The following registers are printed:

- a) BB\_paprd\_ctrl0\_b0
- b) BB\_paprd\_ctrl1\_b0
- c) BB\_paprd\_ctrl2\_b0
- d) paprd\_enable\_0
- e) paprd\_adaptive\_am2pm\_enable\_0
- f) paprd\_valid\_pa\_setting\_0

There are many other registers with PAPRD relevant data. All these registers are logged. If it is necessary to reduce logs in the future, such a reduction can be done based on level of information required. Also, DPD critical information is logged as follows:

- a) pPaprdStruct->workingChannel
- b) pPaprdStruct->smState
- c) pPaprdStruct->transmitted\_training\_packets
- d) g\_temperature
- e) coarse\_fine\_idx
- f) initial\_coarse\_fine\_index
- g) pPaprdStruct->bestSQ
- h) pPaprdStruct->txChain
- i) pPaprdStruct->tableIndex
- j) pPaprdStruct->best\_coarse\_idx

### 22.17.3.2 IBF failure

When an IBF failure occurs, five sessions of six retries have occurred. If this threshold is exceeded, IBF failure is signaled. In this scenario, multiple registers and state machine values need to be logged. The following registers are printed:

- PHY\_BB\_IBF\_CTRL\_ADDRESS
- BB\_ibfcal\_rfcn
- BB\_txbf\_implicit\_rx\_ctrl
- BB\_PAPRD\_TRAIN\_AGC0\_ADDRESS
- BB\_PAPRD\_TRAIN\_AGC1\_ADDRESS
- PHY\_BB\_FORCE\_CLOCK\_ADDRESS

There are many other registers with IBF relevant data. All these registers are logged. If it is necessary to reduce logs in the future, such a reduction can be done based on level of information required. Also, the following IBF critical information is logged:

- HW disabling
- pIBFStruct->smState
- pIBFStruct->workingChannel
- IBFInProcess
- DPDInProcess
- pIBFStruct->txChainMask;
- pIBFStruct->rxChainMask;
- ClearToSend value

### 22.17.3.3 Noise floor calibration

Noise floor calibration is performed every 30 seconds. In case of failure, the following registers can be logged:

1. PHY\_BB\_AGC\_CONTROL\_ADDRESS
2. PHY\_BB\_MULTICHAIN\_ENABLE\_ADDRESS
3. PHY\_BB\_FORCE\_SS\_CTRL\_ADDRESS
4. PHY\_BB\_CCA\_B0\_ADDRESS
5. PHY\_BB\_CCA\_B1\_ADDRESS
6. PHY\_BB\_CCA\_B2\_ADDRESS
7. PHY\_BB\_CCA\_B3\_ADDRESS
8. PHY\_BB\_CCA\_CTRL\_2\_B0\_ADDRESS
9. PHY\_BB\_CCA\_CTRL\_2\_B1\_ADDRESS
10. PHY\_BB\_CCA\_CTRL\_2\_B2\_ADDRESS
11. PHY\_BB\_CCA\_CTRL\_2\_B3\_ADDRESS

In addition to these registers, the following variables can also be logged:

1. calInfo->nf.nfHistEnable
2. pResetStruct->nfCalHungCount
3. g\_Original\_waitForNfCalToComplete
4. programListenMode values currently used
5. calInfo->nf.curNfHistChannel
6. pResetStruct->EEPROM\_NfThresh

7. num\_rx\_chains
8. calInfo->nf.nfLimits
9. g\_NoiseFloorVal

These registers and values are not an exhaustive set. There are many more values, which need to be progressively added, depending on the level of debug information required.

## 22.17.4 Identification of non-calibration triggers—Firmware

Apart from the aforementioned calibration failures, the following data needs to be logged anticipating the following mentioned failures.:

- Tx timeout issues – Utility percentage, NF values
- Reset and Channel scan issues – Reset cause and Reset type (WARM / COLD / WARM\_RESTORECAL,COLD\_RESTORECAL)

### 22.17.4.1 High-level architecture

In the following diagram, the WLAN logging module and FW logging components are the blocks or entities that are implemented.

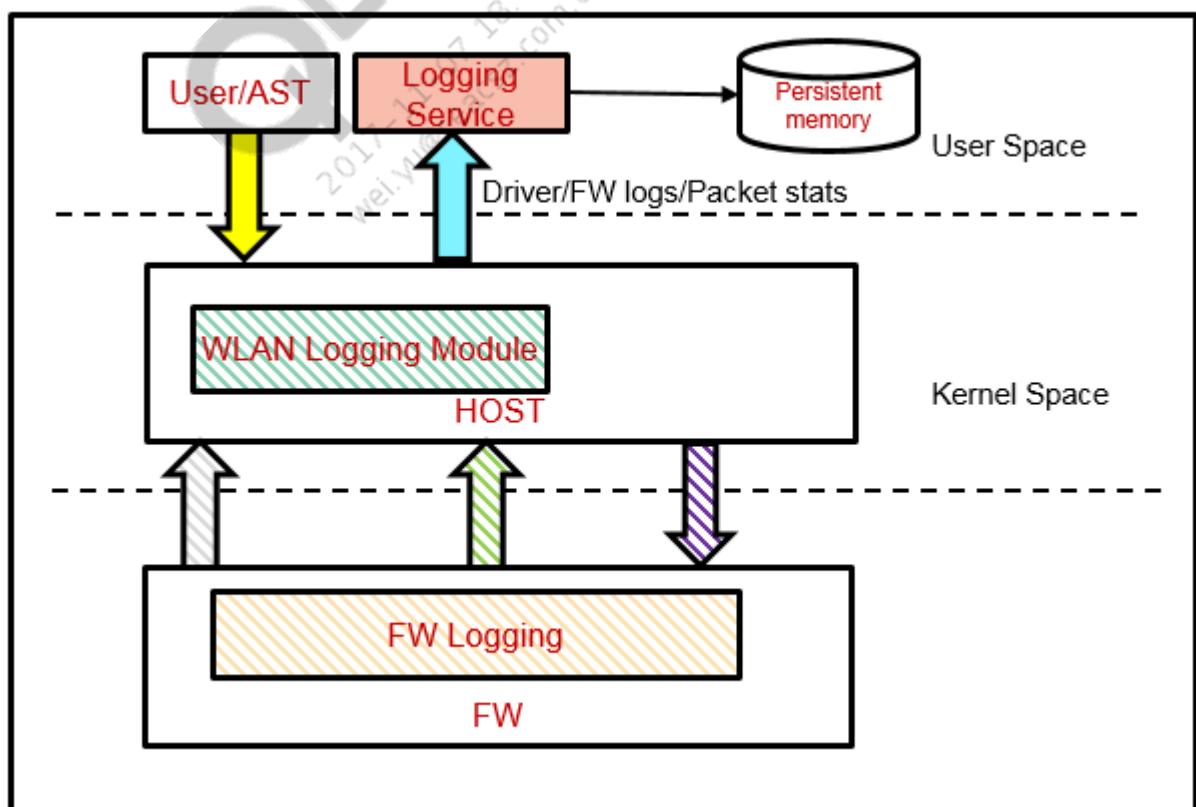


Figure 22-2 High Level Architecture Diagram.

|                                                                                                                                                                |                                                                                                                                                                                                   |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Logging service                                                                                                                                                | Any logging service can keep writing these logs to some internal memory (or) Host can write the logs to some proc or debug entry                                                                  |
| WLAN Logging Module                                                                                                                                            | <ul style="list-style-type: none"> <li>• Trigger based log collection</li> <li>• Collect log info from Host/FW and post to logging service (not for Endive)</li> <li>• Flush Host logs</li> </ul> |
| FW Logging                                                                                                                                                     | <ul style="list-style-type: none"> <li>• Trigger based log collection</li> <li>• Flush FW logs based on Host req.</li> </ul>                                                                      |
| <b>NOTE</b> The following services are planned for future implementation:                                                                                      |                                                                                                                                                                                                   |
| Collected logs posted to logging service                                                                                                                       |                                                                                                                                                                                                   |
| User can configure logging level and log information to be collected on specific error conditions in Host/ FW.                                                 |                                                                                                                                                                                                   |
| Send FW logs to Host through WMI event over control mailbox.                                                                                                   |                                                                                                                                                                                                   |
| Fatal events detected by the FW. Events such as Tx/Rx stuck and inform Host through WMI Event by logging necessary information.                                |                                                                                                                                                                                                   |
| Host can detect fatal events such as connection failures, etc. and request FW to dump the necessary information. Send command to the FW to flush all the logs. |                                                                                                                                                                                                   |

Data collected for smart logs can be saved as a RAR file. This RAR file has a notebook, where details of all registers used in a particular module, and also the possible values to be logged are listed for calibration failures.

## 22.18 WMI support for debug statements in UTF

Until QCA\_Networking\_2017.SPF.5.0 CS release, although support for logging debug statements in mission mode is available, some special rework was necessary for debugging in FTM in the form of firmware debug board with UART support and for IPQ401x chipsets. To simplify debugging, starting with the QCA\_Networking\_2017.SPF.5.0 CSU1 release, the logging of debug statements is extended to factory test mode (FTM).

With this functionality of WMI support for debug print statements in unified test firmware (UTF), DBGLOG\_RECORD\_LOG statements can be added in UTF in the same way as in mission mode so that the debug prints can be viewed on the same console.

The method of operation is the same as in mission mode. The dbglog library is not included for UTF compilation at present; therefore, it has been included for this feature. A new module (FTM\_DEBUG) has been created with new debug IDs (FTM\_DBGID\_WMI\_INIT\_RECV and FTM\_DBGID\_SETUP\_CHANNEL). The existing dbglog APIs are used for initializing the infrastructure and logging the debug messages. An event has been created in UTF-specific to dbglog framework to send the buffers with the debug messages to the host. When the buffer becomes full or when the timer expires, an event is sent to the host to dump the messages in the buffer.

With the feature enabled, certain delay is observed because of message logging and sending buffers to host. Subsequently, a slight delay occurs in the responses from UTF and therefore, a few tests might fail. For example, a few iterations of Rx PER and EVM tests might fail. It is necessary to adjust the time limits accordingly. For example, if Rx PER test fails, the developer must increase the value of the “msecDelaybeforePktSend” parameter in the QSPR test tree.

The FTM debug module can be turned on / off with the following commands. The debug module number for FTM is 37.

```
iwpriv wifiX dl_modon 37
iwpriv wifiX dl_modoff 37
```

## 22.19 Display latency for each PPDU using packet log tool

Packet log decoder script has been enhanced to have latency information for each PPDU. Timestamp of PPDU posted to hw queue is logged. After receiving the Tx completion response from hardware for a particular PPDU, Schedule start time and schedule end time of a successive PPDU is captured and logged. These timestamps are displayed when packet log decoder is used to decode the packet log dump files.

Before this enhancement was implemented for the packet log decoder script, Tx rate control statistics displayed counters to signify MCS rate and number of chains only. Tx rate control statistics are enhanced to display bandwidth information, in addition to MCS rate/no. of chains.

With the Pktlog -R option, bandwidth information is displayed only for Tx side. For Rx stats, bandwidth information is not displayed.

## 22.19.1 Latency information

Run the following command to display all the records from packet log dump.

```
Pktlogconf -e -a wifix -s 6000000
```

After few seconds, stop pktlog script using the following command.

```
Pktlogconf -d wifix
```

Copy the packet log dump (proc file – size appears as zero but not empty file) file from the following path.

```
Cp /proc/ath_pktlog/wifix ~/pktlog.dat
```

Use the following command in the Linux workstation to decode dump file and display records.

```
Perl packetlogdecoder.pl -a pktlog.dat
```

If the grep operation is performed with the output, the timestamp entries related to latency feature are noticed as follows.

```
:TX_RECTYPE_PPDU_TIMESTAMP : sched_start_time:122923190 sched_end_time:122924008 num_frames:44
```

```
:TX_RECTYPE_TX_SCHED_POST_TIMESTAMP : post_time:122924330
```

```
:TX_RECTYPE_PPDU_TIMESTAMP : sched_start_time:122924338 sched_end_time:122925682 num_frames:73
```

```
:TX_RECTYPE_TX_SCHED_POST_TIMESTAMP : post_time:122925881
```

```
:TX_RECTYPE_PPDU_TIMESTAMP : sched_start_time:122925889 sched_end_time:122927805 num_frames:95
```

```
:TX_RECTYPE_TX_SCHED_POST_TIMESTAMP : post_time:122928002
```

## 22.19.2 Pktlog -R

Run the following command to display all the records from packet log dump:

```
Pktlogconf -e -a wifix -s 6000000
```

Run the following command after a few seconds to stop pktlog script:

```
Pktlogconf -d wifix
```

Copy the packet log dump (proc file – size appears as zero but not empty file) file from the following path:

```
Cp /proc/ath_pktlog/wifix ~/pktlog.dat
```

If the grep operation is performed with the output, the timestamp entries related to latency feature are noticed as follows.

```
Perl packetlogdecoder.pl -R pktlog.dat
```

The output is shown as follows:

```
-- Tx Rate Statistics --
      Num      Avg #          Subframes Failures    PER      Excess Failures
Excess Failures %          Rate     Frames Subfr   %      Good     Bad
Rate           Data          RTS      Data
-----  

-----  

48x1 -160Mhz    219  54.8  85.82%    12008  0  0.00%    0  0
0.00% 0.00%
48x1 -40Mhz      2   48.0  0.69%      96  0  0.00%    0  0
0.00% 0.00%
01Lx1 -20Mhz     19  50.9  6.92%      968  0  0.00%    0  0
0.00% 0.00%
M04x1 -20Mhz     3   45.3  0.97%      136  0  0.00%    0  0
0.00% 0.00%
M05x1 -20Mhz     1   56.0  0.40%      56  0  0.00%    0  0
0.00% 0.00%
M07x1h-20Mhz    13  56.0  5.20%      728  0  0.00%    0  0
0.00% 0.00%
-----  

-----  

Total Data frames : 13992
Total RTS failures : 0 (PER = 0.00%)
Total Data failures : 0 (PER = 0.00%)
Note: Excessive Retries (ERs) are accounted for by the last rate of the series.
```

### 22.19.3 Sample test scenario

The following configuration file can be used to configure in maximum VAP mode to test per-VAP statistics.

#### 22.19.3.1 AP Setup

```
uci set wireless.wifi0=wifi-device
uci set wireless.wifi0.type=qcawifi
uci set wireless.wifi0.macaddr=8C:FD:02:00:c9:c9
uci set wireless.wifi0.hwmode=11ac
uci set wireless.wifi0.disabled=0
uci set wireless.wifi0.htmode=HT80
uci set wireless.wifi0.channel=100
uci set wireless.wifi0.txchainmask=15
uci set wireless.wifi0.rxchainmask=15
uci set wireless.wifi0.mode=ap
uci set wireless.@wifi-iface[0]=wifi-iface
uci set wireless.@wifi-iface[0].device=wifi0
uci set wireless.@wifi-iface[0].network=lan
uci set wireless.@wifi-iface[0].mode=ap
uci set wireless.@wifi-iface[0].ssid=5g_wrt
uci commit wireless
```

### 22.19.3.2 STA1 Setup

```

uci set wireless.wifi0=wifi-device
uci set wireless.wifi0.type=qcawifi
uci set wireless.wifi0.macaddr=84:60:aa:00:c9:cc
uci set wireless.wifi0.hwmode=11ac
uci set wireless.wifi0.disabled=0
uci set wireless.wifi0.htmode=HT80
uci set wireless.wifi0.channel=100
uci set wireless.wifi0.txchainmask=15
uci set wireless.wifi0.rxchainmask=15
uci set wireless.wifi0.mode=sta
uci set wireless.@wifi-iface[0]=wifi-iface
uci set wireless.@wifi-iface[0].device=wifi0
uci set wireless.@wifi-iface[0].network=lan
uci set wireless.@wifi-iface[0].mode=sta
uci set wireless.@wifi-iface[0].wds=0
uci set wireless.@wifi-iface[0].ssid=5g_wrt

uci commit wireless

```

### 22.19.3.3 STA2 Setup

```

uci set wireless.wifi0=wifi-device
uci set wireless.wifi0.type=qcawifi
uci set wireless.wifi0.macaddr=84:60:aa:00:c9:d3
uci set wireless.wifi0.hwmode=11ac
uci set wireless.wifi0.disabled=0
uci set wireless.wifi0.htmode=HT80
uci set wireless.wifi0.channel=100
uci set wireless.wifi0.txchainmask=15
uci set wireless.wifi0.rxchainmask=15
uci set wireless.wifi0.mode=sta
uci set wireless.@wifi-iface[0]=wifi-iface
uci set wireless.@wifi-iface[0].device=wifi0
uci set wireless.@wifi-iface[0].network=lan
uci set wireless.@wifi-iface[0].mode=sta
uci set wireless.@wifi-iface[0].wds=0
uci set wireless.@wifi-iface[0].ssid=5g_wrt
uci commit wireless

```

### 22.19.3.4 Configure IP Address

After AP and STA are associated, do the following on STA1:

```

ifconfig br-lan 0.0.0.0
ifconfig ath0 192.168.1.10

```

After AP and STA are associated, do the following on STA1:

```

ifconfig br-lan 0.0.0.0
ifconfig ath0 192.168.1.11

```

### 22.19.3.5 UDP Traffic generation – AP > STA1 and AP > STA2

Execute from STA1: *iperf -u -s -i 1*

Execute from AP:

```
iperf -c 192.168.1.10 -u -b 1000M -i 1 -t 10000
```

818Mbps

Execute from STA2:

```
iperf -u -s -i 1
```

Execute from AP:

```
iperf -c 192.168.1.11 -u -b 1000M -i 1 -t 10000
```

818Mbps

### 22.19.3.6 Results

Packet log for latency and –R option displays the following sample output.

If the grep operation is performed with the output, the time stamp entries related to latency feature are noticed as follows.

```
:TX_RECTYPE_PPDU_TIMESTAMP : sched_start_time:122923190 sched_end_
time:122924008 num_frames:44
:TX_RECTYPE_TX_SCHED_POST_TIMESTAMP : post_time:122924330
:TX_RECTYPE_PPDU_TIMESTAMP : sched_start_time:122924338 sched_end_
time:122925682 num_frames:73
:TX_RECTYPE_TX_SCHED_POST_TIMESTAMP : post_time:122925881
:TX_RECTYPE_PPDU_TIMESTAMP : sched_start_time:122925889 sched_end_
time:122927805 num_frames:95
:TX_RECTYPE_TX_SCHED_POST_TIMESTAMP : post_time:122928002
```

Packet log –R option displays fields as shown in the following output:

-- Tx Rate Statistics --

| Excess Failures | Rate         | RTS   | Data  | Num    | Avg # | Subframes |       | Failures | PER   | Excess Failures |      |
|-----------------|--------------|-------|-------|--------|-------|-----------|-------|----------|-------|-----------------|------|
|                 |              |       |       | Frames | Subfr | %         | Good  | Bad      |       | RTS             | Data |
|                 |              |       |       |        |       |           |       |          |       |                 |      |
| 0.00%           | 48x1 -160Mhz | 0.00% | 0.00% | 219    | 54.8  | 85.82%    | 12008 | 0        | 0.00% | 0               | 0    |
| 0.00%           | 48x1 -40Mhz  | 0.00% | 0.00% | 2      | 48.0  | 0.69%     | 96    | 0        | 0.00% | 0               | 0    |
| 0.00%           | 01Lx1 -20Mhz | 0.00% | 0.00% | 19     | 50.9  | 6.92%     | 968   | 0        | 0.00% | 0               | 0    |

```

M04x1 -20Mhz      3   45.3   0.97%     136   0    0.00%   0   0
0.00% 0.00%
M05x1 -20Mhz      1   56.0   0.40%      56   0    0.00%   0   0
0.00% 0.00%
M07x1h-20Mhz     13   56.0   5.20%     728   0    0.00%   0   0
0.00% 0.00%
-----
-----
Total Data frames : 13992
Total RTS failures: 0 (PER = 0.00%)
Total Data failures: 0 (PER = 0.00%)
Note: Excessive Retries (ERs) are accounted for by the last rate of the series.

```

## 22.20 Enhanced TID queue statistics

The following enhancements are introduced for per-peer TID queue statistics in the output of the iwpriv wifi1 fc\_peer\_stats <mac\_address>:

- Addition of association ID
- Modification of byte\_cnt from a hexadecimal value to a decimal value
- Addition of high\_watermark for each queue (the largest queue length a queue has seen)
- Addition of per-client Tx\_Desc\_fail

The following is an example of the output of the command with the fc\_peer\_stats parameter with the additional fields. The statistics are cleared after print on console every time.

```

root@OpenWrt:~# iwpriv wifi0 fc_peer_stats 8c:fd:f0:01:05:1f
[ 207.553264] Found peer_id=1
[ 207.555482] ----- per-Peer/TIDQ Stats -----
[ 207.561168] ----- Assoc ID = 1 -----
[ 207.566291] -----
[ 207.568853] peer_id 1, tid 0
[ 207.572664] -----+
[ 207.575445] byte_cnt | 255
[ 207.578881] queue_len | 3961
[ 207.581505] max_queue_len | 4096
[ 207.585879] high_watermark| 4096
[ 207.588284] Enqueue_Cnt : 277905
[ 207.592564] Dequeue_Cnt : 278228
[ 207.595563] Dequeue_Bytectn : 420680736
[ 207.599000] Enqueue_Bytectn : 420635376
[ 207.603842] Dequeue_Req_bytectn : 422545500
[ 207.607872] Dequeue_Req_pktn : 501760
[ 207.611277] Peer_q_full : 80757
[ 207.615713] Msdu_TTL_Expiry : 0
[ 207.618025] Hol_Cnt : 0
[ 207.621180] Peer_Tx_Desc_Fail : 68
[ 207.627116] -----
[ 207.629678] peer_id 1, tid 1
[ 207.633489] -----+
[ 207.636082] byte_cnt | 0
[ 207.639643] queue_len | 0

```

```
[ 207.650421] max_queue_len | 4096
[ 207.653577] high_watermark| 0
[ 207.655732] Enqueue_Cnt : 0
[ 207.659231] Dequeue_Cnt : 0
[ 207.661543] Dequeue_Bytetcnt : 0
[ 207.665729] Enqueue_Bytetcnt : 0
[ 207.668072] Dequeue_Req_bytetcnt : 0
[ 207.672696] Dequeue_Req_pktcnt : 0
[ 207.675288] Peer_q_full : 0
[ 207.678819] Msdu_TTL_Expiry : 0
[ 207.681412] Hol_Cnt : 0
[ 207.684504] (Value in TID0 is valid)Peer_Tx_Desc_Fail : 0
[ 207.690159] -----
[ 207.692877] peer_id 1, tid 2
[ 207.696282] -----
[ 207.699125] byte_cnt | 0
[ 207.702717] queue_len | 0
[ 207.705029] max_queue_len | 4096
[ 207.708903] high_watermark| 0
[ 207.711746] Enqueue_Cnt : 0
[ 207.714058] Dequeue_Cnt : 0
[ 207.717525] Dequeue_Bytetcnt : 0
[ 207.720118] Enqueue_Bytetcnt : 0
[ 207.723992] Dequeue_Req_bytetcnt : 0
[ 207.726929] Dequeue_Req_pktcnt : 0
[ 207.731052] Peer_q_full : 0
[ 207.733302] Msdu_TTL_Expiry : 0
[ 207.737050] Hol_Cnt : 0
[ 207.738925] (Value in TID0 is valid)Peer_Tx_Desc_Fail : 0
[ 207.753014] -----
[ 207.756138] peer_id 1, tid 3
[ 207.758512] -----
[ 207.762792] byte_cnt | 0
[ 207.765135] queue_len | 0
[ 207.768447] max_queue_len | 4096
[ 207.771040] high_watermark| 0
[ 207.774351] Enqueue_Cnt : 0
[ 207.777475] Dequeue_Cnt : 0
[ 207.779787] Dequeue_Bytetcnt : 0
[ 207.783661] Enqueue_Bytetcnt : 0
[ 207.786129] Dequeue_Req_bytetcnt : 0
[ 207.790346] Dequeue_Req_pktcnt : 0
[ 207.793595] Peer_q_full : 0
[ 207.796282] Msdu_TTL_Expiry : 0
[ 207.799875] Hol_Cnt : 0
[ 207.801811] (Value in TID0 is valid)Peer_Tx_Desc_Fail : 0
[ 207.808122] -----
[ 207.810934] peer_id 1, tid 4
[ 207.814245] -----
[ 207.817057] byte_cnt | 0
[ 207.820524] queue_len | 0
[ 207.822992] max_queue_len | 4096
[ 207.826835] high_watermark| 0
[ 207.829084] Enqueue_Cnt : 0
[ 207.832552] Dequeue_Cnt : 0
[ 207.834770] Dequeue_Bytetcnt : 0
```

```
[ 207.838644] Enqueue_Byetcnt : 0
[ 207.841205] Dequeue_Req_byetcnt : 0
[ 207.845735] Dequeue_Req_pktcnt : 0
[ 207.848359] Peer_q_full : 0
[ 207.851921] Msdu_TTL_Expiry : 0
[ 207.862074] Hol_Cnt : 0
[ 207.863730] (Value in TID0 is valid)Peer_Tx_Desc_Fail : 0
[ 207.870165] -----
[ 207.873664] peer_id 1, tid 5
[ 207.880943] -----
[ 207.886129] byte_cnt | 0
[ 207.888691] queue_len | 0
[ 207.893908] max_queue_len | 4096
[ 207.898313] high_watermark| 0
[ 207.902655] Enqueue_Cnt : 0
[ 207.905123] Dequeue_Cnt : 0
[ 207.910278] Dequeue_Byetcnt : 0
[ 207.915089] Enqueue_Byetcnt : 0
[ 207.919462] Dequeue_Req_byetcnt : 0
[ 207.925710] Dequeue_Req_pktcnt : 0
[ 207.930771] Peer_q_full : 0
[ 207.934989] Msdu_TTL_Expiry : 0
[ 207.937925] Hol_Cnt : 0
[ 207.942674] (Value in TID0 is valid)Peer_Tx_Desc_Fail : 0
[ 207.950109] -----
[ 207.954857] peer_id 1, tid 6
[ 207.958419] -----
[ 207.963261] byte_cnt | 0
[ 207.967697] queue_len | 0
[ 207.971977] max_queue_len | 4096
[ 207.976413] high_watermark| 0
[ 207.980443] Enqueue_Cnt : 0
[ 207.984629] Dequeue_Cnt : 0
[ 207.988503] Dequeue_Byetcnt : 0
[ 207.991440] Enqueue_Byetcnt : 0
[ 207.996844] Dequeue_Req_byetcnt : 0
[ 208.003811] Dequeue_Req_pktcnt : 0
[ 208.007216] Peer_q_full : 0
[ 208.012558] Msdu_TTL_Expiry : 0
[ 208.017213] Hol_Cnt : 0
[ 208.020524] (Value in TID0 is valid)Peer_Tx_Desc_Fail : 0
[ 208.028678] -----
[ 208.032927] peer_id 1, tid 7
[ 208.037613] -----
[ 208.042205] byte_cnt | 0
[ 208.046141] queue_len | 0
[ 208.050109] max_queue_len | 4096
[ 208.054420] high_watermark| 0
[ 208.056982] Enqueue_Cnt : 0
[ 208.061480] Dequeue_Cnt : 0
[ 208.065448] Dequeue_Byetcnt : 0
[ 208.070040] Enqueue_Byetcnt : 0
[ 208.074445] Dequeue_Req_byetcnt : 0
[ 208.079287] Dequeue_Req_pktcnt : 0
[ 208.085067] Peer_q_full : 0
[ 208.088909] Msdu_TTL_Expiry : 0
```

```
[ 208.093783] Hol_Cnt : 0
[ 208.095782] (Value in TID0 is valid)Peer_Tx_Desc_Fail : 0
[ 208.103748] -----
```

The per-radio global TID queue statistics is not modified due to this enhancement. The following is an example of the output of the iwpriv wifiN fc\_stats\_global command:

```
root@OpenWrt:~# iwpriv wifi0 fc_stats_global
[ 214.105248] ----- Global Stats -----
[ 214.109871] HTTFetch ==>: 8183
[ 214.112902] HTTFetch_NoId ==>: 3581
[ 214.115745] HTTFetch_Conf ==>: 2
[ 214.119525] HTTDesc_Alloc ==>: 8185
[ 214.122430] HTTDesc_Alloc_Fail ==>: 0
[ 214.126772] HTTFetch_Resp ==>: 8187
[ 214.129740] Dequeue_Cnt ==>: 810783
[ 214.133270] Enqueue_Cnt ==>: 814880
[ 214.137082] Queue_Bypass_Cnt ==>: 1190
[ 214.140830] Enqueue_Fail_Cnt ==>: 0
[ 214.143705] Dequeue_Bytetcnt ==>: 1226430072
[ 214.148391] Enqueue_Bytetcnt ==>: 1232624736
[ 214.152702] Peer_q_Full ==>: 196859
[ 214.155420] Dequeue_Req_Bytetcnt ==>: 1234920375
[ 214.161012] Dequeue_Req_Cnt ==>: 1525440
[ 214.164167] Queue_Depth ==>: 4097
[ 214.168041] Tx_Compl_ind ==>: 1168120
[ 214.171602] Tx_Desc_fail ==>: 0
[ 214.174195] Intr_Taskletbin500 ==>: 0
[ 214.178381] Intr_Taskletbin1000 ==>: 0
[ 214.181599] Intr_Taskletbin2000 ==>: 0
[ 214.194001] Intr_Taskletbin4000 ==>: 0
[ 214.197438] Intr_Taskletbin6000 ==>: 0
[ 214.200718] Intr_Taskletbinhigh ==>: 0
[ 214.204967] HTTFetchbin500 ==>: 0
[ 214.207997] HTTFetchbin1000 ==>: 0
[ 214.211996] HTTFetchbin2000 ==>: 0
[ 214.214682] HTTFetchbinhigh ==>: 0
[ 214.218431] Queue_Occupancy ==>: 0
[ 214.221337] Queue_Occupancy_Bin2 ==>: 0
[ 214.225866] Queue_Occupancy_Bin8 ==>: 0
[ 214.229490] Queue_Occupancy_Bin16 ==>: 0
[ 214.233020] Queue_Occupancy_Bin32 ==>: 0
[ 214.237769] Queue_Occupancy_Binhigh ==>: 0
wifi0      fc_stats_global:4308
```

## 22.21 Enhanced client statistics for acknowledgment failures, modulation rates, authentication details

This section describes the display of enhanced statistics of acknowledgment failures, modulation rates, authentication details. Use the apstats -s -m xx:xx:xx:xx:xx:xx (MAC address of station) command, the athstats command, the wlanconfig ath1 list command, and the iwpriv ath1 txrx\_fw\_stats 1 command, appropriately, to view these enhanced client or STA statistical details. The following additional statistics are displayed:

**1. tx unaggregated unacked frames, tx aggr unacked subframes, and MPDUs\_ack\_failed—**

Number of unacknowledged data MPDU transmissions. Firmware transmits this statistical counter by enabling the packetlog event, WMI\_PKTLOG\_EVENT\_TX, for QCA9880 and as a part of host target transport (HTT) statistics for QCA9980. This statistic was previously available for display for QCA9980 family of chipsets, and this implementation is extended for displaying values for QCA9880 chipsets.

**NOTE** These values are counts based for offload and DA.

For direct attach (DA) models, the *tx unaggregated unacked frames, tx aggr unacked subframes* fields are displayed in the output of the `athstats -i wifi` command. For offload (OL) models, the *MPDUs\_ack\_failed* field is displayed in the output of the `iwpriv athX txrx_fw_stats 1` command.

**NOTE** For offload models, enter the `iwpriv wifi enable_ol_stats 1` command to enable the collection of offload statistics before viewing this parameter in the output of the `iwpriv athX txrx_fw_stats 1` command.

**2. Average Tx Rate (kbps)—**Weighted average of the Tx modulation rate for data MPDU.

Firmware transmits this information when WMI\_PKTLOG\_EVENT\_RCU is enabled for QCA9880 and as a part of enhanced statistics for QCA9980. The calculated average Tx modulation rate value is updated to display in apstats tool.

The *Average Tx Rate (kbps)* field is displayed in the output of the `apstats -s -m <mac_addr>` command for DA radios. The *Avg ppdu Tx Rate (kbps)* field is displayed in the output of the `apstats -s -m <mac_addr>` command for OL radios.

**NOTE** For offload models, enter the `iwpriv wifi enable_ol_stats 1` command to enable the collection of offload statistics before viewing this parameter in the output of the `apstats -s -m <mac_addr>` command.

**3. Average Rx Rate (kbps)—**Weighted average of the Rx modulation rate for data MPDU. This information is retrieved from Rx descriptors and handled in rx\_indication\_msg. No separate path is implemented for QCA9880 or QCA9980 per design. The calculated average Rx modulation rate value is updated to display in apstats tool.

The *Average Rx Rate (kbps)* field is displayed in the output of the `apstats -s -m <mac_addr>` command for DA radios. The *Avg ppdu Rx Rate (kbps)* field is displayed in the output of the `apstats -s -m <mac_addr>` command for OL radios.

**NOTE** For offload models, enter the `iwpriv wifi enable_ol_stats 1` command to enable the collection of offload statistics before viewing this parameter in the output of the `apstats -s -m <mac_addr>` command.

**4. Channel Busy—**Percentage of each second for which the channel is busy (TX/RX/noise).

This value is already displayed in the output of the `apstats` command in the Channel Utilization field. The channel utilization parameter is disabled by default; it can be enabled by the `iwpriv athx chutil_enab 1` command.

The *value* displayed in the output of the `iwpriv athx get_chutil` command represents the minimum, maximum, and average over measurement intervals.

5. **802.11 Auth Attempts**—Number of user equipment/UE authentication attempts. This parameter displays the values that the WLAN driver calculates per VAP and per radio in the output of the `apstats` command.

The *802.11 Auth Attempts* field is displayed in the output of the `apstats -r -i wifiX` command.

6. **MLME Authorize Success**—Number of UE authentications for MAC layer management entity (MLME). This parameter considers the number of UEs that are in MLME authorized state. This value is calculated as the number of authorized successes and displayed in the output of the `apstats` command for VAP-level and radio-level statistics.

The *MLME Authorize Success* field is displayed in the output of the `apstats -r -i wifiX` command.

7. **Self BSS chan util, OBSS chan util** (Busy by Other Radio, Busy by non-WiFi)—Percentage of each second for which the transmitter is busy by another radio and the percentage of each second for which the transmitter is busy by non-Wi-Fi. Both these types of percentages are not separated and calculated by hardware counters. As a result, the combined value of (Busy by other radio + Busy by non-Wi-Fi) and Self-BSS are displayed in the output of the `apstats -r -i wifiX` command as per-radio statistics.

These *Self BSS chan util, OBSS chan util* parameters apply per-radio and not per-VAP, and are displayed in the output of the `apstats -r -i wifiX` command.

8. **ASSOCTIME**—Total amount of time for which the clients are connected. This parameter is per radio, per AP, and per SSID.

The *ASSOCTIME* parameter is also displayed in the output of the `wlanconfig athx list` command.

9. **802.11 Auth Attempts, 802.11 Auth Success**—Number of user equipment/UE (client) authentication failures. This value is calculated as follows:

$$\text{Authentication failures} = \text{Authentication Attempts} - \text{Authentication Successes} \quad (1)$$

This value is displayed in `apstats` tool in both radio-level and VAP-level statistics.

The *802.11 Auth Attempts, 802.11 Auth Success* fields are displayed in the output of the `apstats -r -i wifiX` command.

10. **Connections refuse Radio limit**—Number of connection that are rejected because the connected users limit is exceeded. Number of clients that are rejected after the radio limit (`num_vaps`) and vap limit (`max_aid`) are exceeded is updated from driver when node join occurs. This value is displayed in `apstats` tool in both radio and VAP-level statistics. Also, a `iwpriv` command has been added to configure the radio limit dynamically that was earlier hardcoded to a predefined value during attach. Enter the `iwpriv wifiX max_sta <val>` command to configure the maximum number of client connections, where `<val>` is max number of connections allowed per radio (limit) +1.

The *Connection refuse Radio limit* field is displayed in the output of the `apstats -r -i wifiX` command.

11. **AID**—A unique session ID that might be common for more than one record (long). The unique session ID is considered as the association ID (AID).

This *Session ID* parameter is displayed in the output of the `wlanconfig athx list` command for both DA and OL models.

**NOTE** The acknowledgment failures, Tx average modulation rate, and Rx average modulation rate parameters have different path implementation for direct attach and offload architectures. For offload models only, enter the `iwpriv wifix enable_ol_stats 1` command to enable the collection of offload statistics before viewing these parameters in the output of the `apstats` command. This behavior is because these parameters are retrieved from firmware through resource-intensive data path, and also a slight performance impact is observed when these parameters are collected and displayed.

The following sample outputs illustrate the newly added display fields:

#### Output of `apstats` for DA radios:

```
root@OpenWrt:/# apstats -r -i wifi0
Radio Level Stats: wifi0
Tx Data Packets          = 0
Tx Data Bytes            = 0
Rx Data Packets          = 376
Rx Data Bytes            = 59264
Tx Unicast Data Packets = 0
Tx Multi/Broadcast Data Packets = 0
Tx Data Packets per AC:
  Best effort           = 0
  Background            = 0
  Video                 = 0
  Voice                 = 0
Rx Data Packets per AC:
  Best effort           = 0
  Background            = 0
  Video                 = 0
  Voice                 = 0
Channel Utilization (0-255) = <DISABLED>
Tx Beacon Frames         = 1266
Tx Mgmt Frames           = 1619
Rx Mgmt Frames           = 14004
Tx Ctl Frames            = 0
Rx Ctl Frames            = 0
Rx RSSI                  = 41
Rx PHY errors             = 710
Rx CRC errors             = 583
Rx MIC errors             = 0
Rx Decryption errors      = 0
Rx errors                 = 0
Tx failures               = 0
Tx Dropped                = 2
Connections refuse Radio limit = 0
Connections refuse Vap limit = 0
802.11 Auth Attempts     = 1
802.11 Auth Success       = 1
MLME Authorize Attempts   = 1
MLME Authorize Success     = 1
Self BSS chan util        = 16
```

```

OBSS chan util          = 80
Throughput (kbps)       = <DISABLED>
PER over configured period (%) = <DISABLED>
Total PER (%)           = 0

```

### Output of apstats for OL radios:

```

root@OpenWrt:/# apstats -s -m 04:1b:6d:ba:40:29
Node Level Stats: 04:1b:6d:ba:40:29 (under VAP ath1)
Tx Data Packets          = 0
Tx Data Bytes             = 0
Tx Success Data Packets   = 0
Tx Success Data Bytes     = 0
Tx Data Packets per AC:
  Best effort              = 0
  Background                = 0
  Video                      = 0
  Voice                      = 0
Rx Data Packets per AC:
  Best effort              = 0
  Background                = 0
  Video                      = 0
  Voice                      = 0
Tx Success Unicast Data Packets = 0
Tx Success Unicast Data Bytes = 0
Tx Success Multicast Data Packets = 0
Tx Success Multicast Data Bytes = 0
Last Packet Error Rate (PER)    = 13
Rx Data Packets              = 0
Rx Data Bytes                 = 0
Rx Unicast Data Packets      = 0
Rx Unicast Data Bytes        = 0
Rx Multicast Data Packets    = 0
Rx Multicast Data Bytes      = 0
Avg ppdu Tx Rate (kbps)       = 0
Avg ppdu Rx Rate (kbps)       = 0
Last tx rate                  = 175500
Last rx rate                  = 325000
Last mgmt rx rate            = 6000
Rx MIC Errors                 = 0
Rx Decryption errors          = 0
Rx errors                      = 0
Packets Queued                 = 0
Host Discard                   = 0
Tx Mgmt Packets                 = 0
Rx Mgmt Packets                 = 0
Rx mpdu count                   = 0
Rx ppdu count                   = 0
Rx retry count                   = 0
Tx failures                      = 0
Rx RSSI                         = 24
Rx MGMT RSSI                     = 18
Excessive retries per AC:
  Best effort                  = 1170
  Background                    = 0
  Video                        = 0

```

```

Voice = 11
Ack RSSI chain 1 = 0
Ack RSSI chain 2 = 0
Ack RSSI chain 3 = 0
Ack RSSI chain 4 = 0
Band Width = 80
stbc tx(0) rx(0)
chainmask (NSS) tx(1) rx(1)
Tx packets for last one second = 0
Tx bytes for last one second = 0
Rx packets for last one second = 0
Rx bytes for last one second = 0

```

```

root@OpenWrt:/# athstats -i wifi0 | grep unacked
    277 tx unaggregated unacked frames
        0 tx aggr unacked subframes
        0 tx rifs: unacked subframes

```

```

root@OpenWrt:/# wlanconfig ath1 list
ADDR          AID CHAN TXRATE RXRATE RSSI MINRSSI MAXRSSI IDLE TXSEQ
RXSEQ  CAPS      ACAPS     ERP      STATE MAXRATE(DOT11) HTCAPS ASSOCTIME
IES   MODE          PSMODE RXNSS TXNSS
04:1b:6d:ba:40:29   1  149   0M   0M   27     0       27   0   0
65535      E      0         b           0
WME IEEE80211_MODE_11AC_VHT80  0       1       1

```

```

root@OpenWrt:/# apstats -s -m 04:1b:6d:ba:40:29
Node Level Stats: 04:1b:6d:ba:40:29 (under VAP ath0)
Tx Data Packets = 37554
Tx Data Bytes = 49797250
Tx Success Data Packets = 0
Tx Success Data Bytes = 0
Tx Data Packets per AC:
    Best effort = 0
    Background = 0
    Video = 0
    Voice = 0
Rx Data Packets per AC:
    Best effort = 0
    Background = 0
    Video = 0
    Voice = 0
Tx Success Unicast Data Packets = 0
Tx Success Unicast Data Bytes = 0
Tx Success Multicast Data Packets = 0
Tx Success Multicast Data Bytes = 0
Last Packet Error Rate (PER) = 0
Rx Data Packets = 42722
Rx Data Bytes = 39231993
Rx Unicast Data Packets = 42682
Rx Unicast Data Bytes = 39227006
Rx Multicast Data Packets = 40
Rx Multicast Data Bytes = 4987
Average Tx Rate (kbps) = 71247

```

```

Average Rx Rate (kbps)          = 71686
Last tx rate                   = 0
Last rx rate                   = 0
Last mgmt rx rate             = 0
Rx MIC Errors                 = 0
Rx Decryption errors          = 0
Rx errors                      = 0
Packets Queued                = 37554
Host Discard                  = 0
Tx Mgmt Packets               = 0
Rx Mgmt Packets               = 0
Rx mpdu count                 = 0
Rx ppdu count                 = 0
Rx retry count                = 0
Tx failures                    = 0
Rx RSSI                        = 64
Rx MGMT RSSI                  = 0
Excessive retries per AC:
    Best effort                = 0
    Background                 = 0
    Video                       = 0
    Voice                       = 0
    Ack RSSI chain 1            = 0
    Ack RSSI chain 2            = 0
    Ack RSSI chain 3            = 0
    Ack RSSI chain 4            = 0
    Band Width                 = 20
    stbc                        tx(0) rx(0)
    chainmask (NSS)             tx(1) rx(1)
Tx packets for last one second = 0
Tx bytes for last one second  = 0
Rx packets for last one second = 0
Rx bytes for last one second  = 0

```

```

root@OpenWrt:/# iwpriv ath1 txrx_fw_stats 1
[ 1811.510000] ### HOST MSDU TTL Stats ###
[ 1811.510000] Host_msdu_ttl      :0
[ 1811.520000] WAL Pdev stats:
[ 1811.530000]
[ 1811.530000] ### MEM ###
[ 1811.530000] ##### Free memory#####
[ 1811.540000] IRAM Remaining: 1836
[ 1811.540000] DRAM Remaining: 540
[ 1811.550000] SRAM Remaining: 0
[ 1811.550000]
[ 1811.550000] ### Tx ###
[ 1811.560000] comp_queued       :1995897
[ 1811.560000] comp_delivered    :1995897
[ 1811.560000] msdu_enqueued     :2004696
[ 1811.570000] wmm_drop          :0
[ 1811.570000] local_enqueued    :8716
[ 1811.580000] local_freed       :8716
[ 1811.580000] hw_queued         :200322
[ 1811.590000] hw_reaped         :200320
[ 1811.590000] mac_underrun     :0
[ 1811.590000] phy_underrun     :0

```

```
[ 1811.600000] hw_paused :0
[ 1811.600000] seq_posted :0
[ 1811.610000] mu_seq_posted :0
[ 1811.610000] seq_failed :0
[ 1811.620000] seq_restarted :0
[ 1811.620000] tx_abort :0
[ 1811.620000] mpdus_requeued :129736
[ 1811.630000] mpdus_sw_flush :0
[ 1811.630000] mpdus_hw_filter :0
[ 1811.640000] mpdus_truncated :0
[ 1811.640000] mpdus_ack_failed :0
[ 1811.640000] mpdus_expired :0
[ 1811.650000] excess retries :6399
[ 1811.650000] last rc :196
[ 1811.660000] sched self trig :3636
[ 1811.660000] ampdu retry failed:10
[ 1811.660000] illegal rate errs :0
[ 1811.670000] pdev cont xretry :0
[ 1811.670000] pdev tx timeout :0
[ 1811.680000] pdev resets :1
[ 1811.680000] ppdu txop ovf :0
[ 1811.680000] mcast Drop :0
[ 1811.690000]
[ 1811.690000] ### Rx ###
[ 1811.690000] ppdu_route_change :123
[ 1811.700000] status_rcvd :1562499
[ 1811.700000] r0_frags :0
[ 1811.700000] r1_frags :45
[ 1811.710000] r2_frags :55
[ 1811.710000] htt_msdu :1562499
[ 1811.720000] htt_mpdus :1562497
[ 1811.720000] loc_msdu :17697
[ 1811.730000] loc_mpdus :17697
[ 1811.730000] oversize_amsdu :0
[ 1811.740000] phy_errs :0
[ 1811.740000] phy_errs_dropped :18
[ 1811.740000] mpdu_errs :11376
[ 1811.750000] pdev_rx_timeout :0
[ 1811.750000] ovfl_mpdu_errs :0
```

## 22.21.1 Station connection parameters

**Table 22-4 Station connection parameters**

| Parameter              | Command                          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| max_radio_sta<br><val> | iwpriv wifix max_radio_sta <val> | <p>Configure the maximum number of client connections, where &lt;val&gt; is max number of connections allowed per radio (limit) +1. This iwpriv command has been added to configure the radio limit dynamically that was earlier hardcoded to a predefined value during attach.</p> <p>The <i>Connections refuse Radio limit</i> field is displayed in the output of the apstats -r -i wifix command, which indicates the number of connection that are rejected because the connected users limit is exceeded. Number of clients that are rejected after the radio limit (num_vaps) and vap limit (max_aid) are exceeded is updated from driver when node join occurs. This value is displayed in apstats tool in both radio and VAP-level statistics.</p> |

QUALCOMM  
2017-11-07 18:46:38 PST  
wei.yu@ceacs2.com.cn