RALINK TECHNOLOGY, CORP.

RALINK DRIVER DESIGN SPECIFICATIONS

VERSION 0.14 03. 05. 2009

Revision History

Date	Revision	Author	Description
02. 17, 2009	First 0.10	Sample Lin	Initial draft. for RALINK WLAN Station Driver Design Spec.
02. 18, 2009	0.11	Jay Hung	Revise main Probe Function rt28xx_probe() flow chart and description, mainly specify USB interface information Add 3.5 USB block
02. 19, 2009	0.12	WY Juang	Add IEEE802.11s Mesh Module Spec.
03. 05, 2009	0.13	Albert Yang	Add IEEE802.1x Daemon Module Spec.
03. 05, 2009	0.14	Snowpin Lee	Add WPS Module Spec.

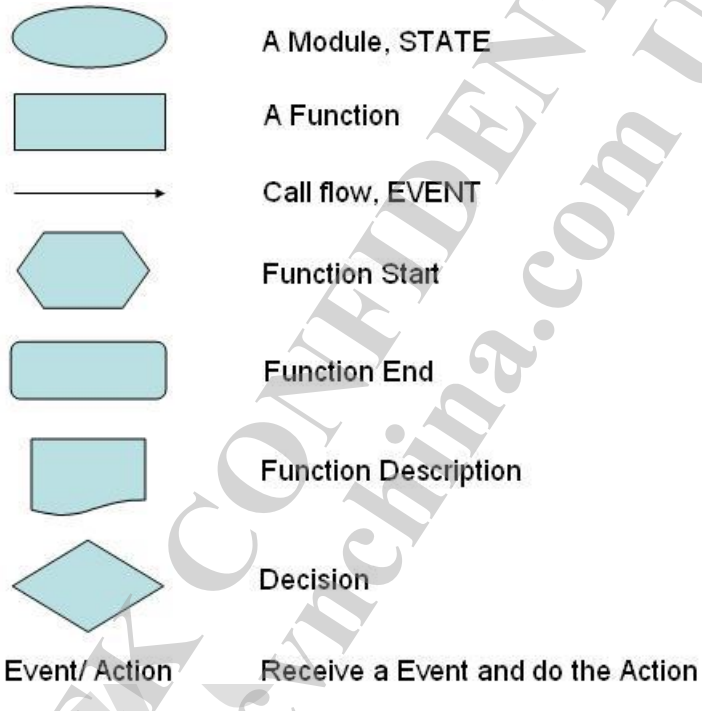
Contents

Figure

1 INTRODUCTION

This document describes the basic design flow (IEEE802.11 & IEEE802.11e & IEEE802.11n) of the RALINK WLAN driver. Specific code patches for different chipsets or module are not included here. They are described in other Module Design Specifications and Release Notes.

1.1 Flow Chart Symbols



1.2 Naming Convention

RT28xx refers to RT2860, RT2870, RT2880, etc. But too many chips are supported, new code base will be renamed to RT_WIFI, means Ralink Technique Wi-Fi.

RT28XX[Function Name]

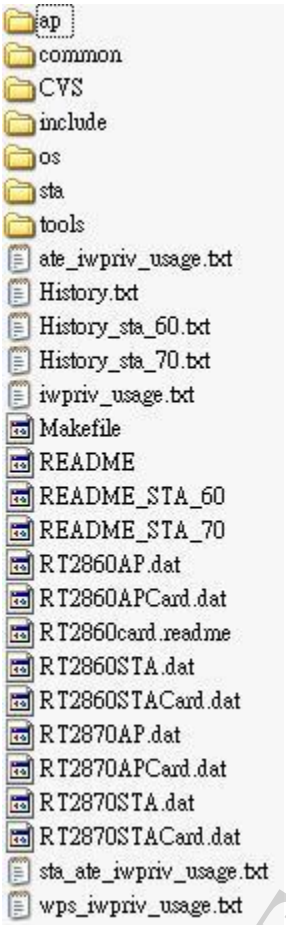
EX1: RT28XXNetDevInit() means there is one dedicated function NetDevInit for PCI device and another one for USB device. You can found one RT28XXNetDevInit() in the 2860_main_dev.c and one in the 2870_main_dev.c

EX2: rt28xx_init() means the initialization function is common for all chipsets.

1.3 Keywords

AP (WLAN Access Point), STA (WLAN Station), FSM (Finite State Machine), API (Application Programming Interface), IOCTL (Input Output Control), MISC (Miscellaneous), MGMT (Management), CTRL (Control), MBSS (Multiple BSS), WDS (Wireless Distribution System), DLS (Direct Link Setup), BBP (Base Band Processor), AUTH (Authentication), ASSOC (Association), BA (Block Ack), AES (Advanced Encryption Standard), TX (Transmission), RX (Receive), MAT (Mac Address Translation), WAPI (Wireless Authentication and Privacy Infrastructure), DFS (Dynamic Frequency Selection), WSC (Wi-Fi Simple Config), EXT (Extension), TBTT (Target Beacon Transmit Time), ATE (Automated Test Equipment), ALLOC (Allocate), INIT (Initialize), MLME (802.11 MAC Layer Management Entity), CONF (Configuration), CHAN (Channel), MCU (Microcontroller Unit), DESCP (Descriptor), AC (Access Category), INT (Interrupt), ISR (Interrupt Service Routine), INFO (Information), RSSI (Received Signal Strength Index), BC/MC (Broadcast/Multicast), Legacy PS (Legacy Power Save), DISASSOC (Disassociate), CMD (Command).

2 FILE STRUCTURE



ap - files used only in AP mode

common - files used in AP and STA modes

cvs - files used by LINUX version control.

include - all header files

os - files used to proved support for different APIs in various operating systems

sta - files used only in STA mode

tools - utilities used for module make

Figure 2-1 RALINK WLAN diver file directory

2.1 Main Directory

The main directory includes the Makefile for the Linux platform, referenced configuration files, the readme file, IOCTL command format, and the version history.

2.2 ap Directory

The ap folder includes all the functions, FSM, Modules, and IOCTL used in AP mode.

2.2.1 MISC

BASIC (ap.c) - Response for AP initialization, AP shutdown, STA entry maintenance, Auto-channel Selection, etc.

Auto-Channel Selection API (ap_autoChSel.c) - Provide related API, such as channel table initialization/insert/deletion etc.

BEACON (ap_connect.c) - Provide all beacon generation/update.

Intrusion Detection (ap_ids.c) - Provide intrusion detection.

RATE SWITCH (ap_mlme.c) - Provide transmission rate switch, handle various event periodically, BBP signal adjusting.

QoS Load (ap_qload.c) - Provide IEEE802.11e QoS Load API.

SANITY CHECK (ap_sanity.c) - Provide all sanity check for request content of PROBE/AUTH/ASSOC request.

2.2.2 IEEE802.11 MLME

PROBE (ap_sync.c) - Response for other AP Beacon listen and STA probe handle.

AUTH (ap_auth.c, ap_authrsp.c) - Response for STA authentication request handle.

ASSOC (ap_sync.c, ap_auth.c, ap_authrsp.c, ap_assoc.c) - Response for STA association request handle.

2.2.3 PACKET

TRANSMISSION/RECEIVE (ap_data.c) - Response for DATA/MGMT/CTRL packet transmission/receive.

2.2.4 SECURITY

Provide security 4-way FSM initialization, WPA/WPA2 handle. (ap_wpa.c)

2.2.5 MODULE

AP CLIENT - Module Initialization, Network Interface Creation/Register/Open/Close. (ap_apcli.c)

Module Body (apcli_assoc.c, apcli_auth.c, apcli_ctrl.c, apcli_sync.c)

MBSS - Module Initialization, Network Interface Creation/Register/Open/Close. (ap_mbss.c)

WDS - Module Initialization, Network Interface Creation/Register/Open/Close. (ap_wds.c)

DLS - Provides Module API. (ap_dls.c)

IEEE802.11r - Provides Module API. (ap_ftkd.c, ap_ftrc.c)

WMM UAPSD - Provides Module API. (ap_uapsd.c)

2.3 common Directory

The common directory includes all the command functions, FSM, and modules used in AP and STA mode.

2.3.1 MISC

ACTION FRAME (action.c) - Provides all management action frame initialization and handle.

BA ACTION (ba_action.c) - Provides IEEE802.11n BA action frame handle.

DATA TX/RX (cmm_data.c) - Provides common management frame transmission, received packets handle, related packet transmission buffer descriptor fill, STA entry reset/insertion/deletion, etc.

IOCTL (cmm_info.c) - Provides various API to get/set WLAN configuration.

SANITY CHECK (cmm_sanity.c) - Provides all sanity check for content of PROBE/AUTH/ASSOC request/response, IEEE802.11n BA Action, IEEE802.11i WPA EAPOL Message, DLS Message.

CHANNEL (cmm_sync.c) - Provides channel list built, channel scan next, etc.

EEPROM (eeprom.c) - Provides EEPROM read/write.

RATE SWITCH and PERIODICAL HANDLE (mlme.c) - Provides rate switch mechanism and periodical hander to handle any periodical event, etc.

RALINK FSM EXT (mlme_ex.c) - Provides another FSM initialization that extends used parameters of old RALINK FSM mechanism.

FIRMWARE (rt2860.bin or rt2870.bin) - Firmware is downloaded into the internal 8051 in initialization stage.

INITIALIZATION (rtmp_init.c) - Provides default BBP/MAC registers, control block allocation, EEPROM setting read, registers initialization, transmission statistics count, 8051 firmware upload, timer utility, LED, etc.

2.3.2 HARDWARE

2.3.2.1 PCI

Provides transmission and receive packet buffer descriptor allocation/free, packet buffer allocation, get packet from receive buffer ring. (2860_rtmp_init.c)

Provides transmission buffer descriptor fill, packet kick out, PCI sleep/awake, radio on/off, etc. (cmm_data_2860.c)

2.3.2.2 USB

Provides transmit and receive packet buffer descriptor allocation/free, packet buffer allocation, get packet from receive buffer ring, thread creation, different transmit queue completion handlers. (2870_rtmp_init.c)

Provides transmission buffer descriptor fill, packet kick out, USB sleep/awake, radio on/off, etc. (cmm_data_2870.c)

Provide related packet transmission/ receive handler by using USB API provided by OS. (rtusb_bulk.c, rtusb_data.c, rtusb_io.c)

2.3.3 PACKET

TRANSMISSION/RECEIVE (ap_data.c) - Response for DATA/MGMT/CTRL packet transmission/receive.

2.3.4 SECURITY

AES - Provides AES API. (cmm_aes.c)

TKIP - Provides TKIP API. (cmm_tkip.c)

MD5 - Provides MD5 API. (md5.c)

SHA2 - Provides SHA2 API. (sha2.c)

SMS4 - Provides SMS4 API. (sms4.c)

HMAC - Provides HMAC API. (hmac.c)

WEP - Provides WEP API. (cmm_wep.c)

WPA - Provides WPA Protocol API. (cmm_wpa.c)

2.3.5 MODULE

WMM ACM - Provides Module API. (acm_comm.c, acm_edca.c, acm_iocl.c)

NET IF BLOCK - Provides Module API. (netif_block.c)

MAT - Provides Module API. (cmm_mat.c, cmm_mat_iparp.c, cmm_mat_ipv6.c, cmm_mat_pppoe.c)

WAPI - Provides Module API. (cmm_wapi.c, wpi_pcrypt.c)

DFS - Provides Module API. (dfs.c, spectrum.c)

WSC - Provides Module API. (dh_key.c, evp_enc.c, wsc.c, wsc_tlv.c)

IEEE802.11r - Provides Module API. (ft_iocl.c)

IGMP SNOOP - Provides Module API. (igmp_snoop.c)

IEEE802.11s MESH - Provides Module API. (mesh.c, mesh_bmpkt.c, mesh_ctrl.c, mesh_forwarding.c, mesh_link_mng.c, mesh_path_mng.c, mesh_sanity.c, mesh_tlv.c)

2.4 include Directory

Includes all header files used in AP and STA mode.

2.5 os Directory

Includes all functions needed to link with kernel supported API, such as timer/task/memory etc.

2.5.1 Linux

2.5.1.1 PCI

Provides device probe handler, module insertion/remove, interrupt enable/disable, transmission/receive tasklet, interrupt handler, PCIe related, etc. (2860_main_dev.c)

2.5.1.2 USB

Provides device probe handler, module insertion/remove, Command thread, Timer thread, TBTT time adjustment, etc. (2870_main_dev.c)

IOCTL (ap_ioctl.c, sta_ioctl.c) - Provides IOCTL for AP and STA.

LINUX COMPILE - Provides various Makefiles for Linux 2.4.x and Linux 2.6.x. (Makefile.x) Provides various Module enable/disable. (config.mk)

ATE - Provides various functions to do ATE.

OS - Provides various virtual layers by calling API supplied from LINUX OS. e.g., timer, packet control, network interface operation, thread, etc.

NETWORK INTERFACE - Provides network interface probe/register/open/close/send/ioctl and chipset init. (rt_main_dev.c)

PROFILE - Provides the profile dat file (e.g., RT2860AP.dat) parsing. (rt_profile.c)

2.6 sta Directory

Includes all functions, FSM, Modules, and IOCTL used in STA mode.

2.6.1 Link

PROBE - Provides related probe functions. (sync.c)

AUTH - Provides related authentication functions. (auth.c)

ASSOC - Provides related association functions. (assoc.c)

FSM - Provides related state machine to do association. (connect.c)

SANITY - Provides sanity check for PROBE/AUTH/ASSOC use. (sanity.c)

2.6.2 Module

DLS - Provides Module API. (dls.c)

LEAP - Provides Module API. (leap.c)

2.6.3 Packet

Provides packet transmission & receive handler. (rtmp_data.c)

2.6.4 Security

Provides related security functions. (wpa.c)

2.7 tools Directory

Includes all utilities, such as bin2h which translates binary file to a header file used in C language.

BIN2H - Provides a tool to translate a binary file to a C header file. (bin2h.c)

3 DRIVER DESIGN

Partition RALINK WLAN driver into five blocks as below:

INIT BLOCK - The block is responsible for device probe, network interface register, software structure/task initialization, latest configuration read, ASIC MAC/ BBP registers initialization, etc.

TRANSMISSION BLOCK - The block is responsible for packet transmission.

RECEIVE BLOCK - The block is responsible for packet receive and delivery to the upper OS layer.

STOP BLOCK - The block is responsible for software structure release, ASIC MAC/ BBP disable, network interface unregister, etc.

IOCTL BLOCK - The block is responsible for configuration change, device command, information read, etc.

USB BLOCK - USB TX Bulk out aggregation, RX Bulk in de-aggregation

OTHER BLOCK - The block is responsible for interrupt handle, MLME task, etc.

3.1 INIT BLOCK

3.1.1 LINUX

3.1.1.1 Device Probe

3.1.1.1.1 PCI

Fill various PCI chip type into rt2860_pci_tbl[].

```
//
// Ralink PCI device table, include all supported chipsets
//
static struct pci_device_id rt2860_pci_tbl[] __devinitdata =
{
    {PCI_DEVICE(NIC_PCI_VENDOR_ID, NIC2860_PCI_DEVICE_ID)},
    {PCI_DEVICE(NIC_PCI_VENDOR_ID, NIC2860_PCIe_DEVICE_ID)},
    {PCI_DEVICE(NIC_PCI_VENDOR_ID, NIC2760_PCI_DEVICE_ID)},
    {PCI_DEVICE(NIC_PCI_VENDOR_ID, NIC2790_PCIe_DEVICE_ID)},
    .
    .
    .
}
```

e.g., NIC_PCI_VENDOR_ID = 0x1814, NIC2860_PCI_DEVICE_ID = 0x0601.

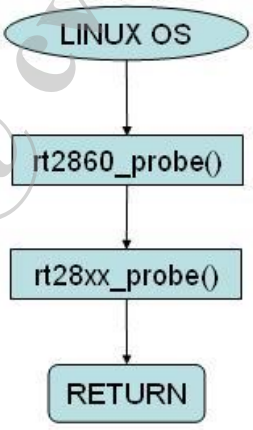


Figure 3-1 PCI INIT flow chart

3.1.1.1.2 USB

Fill various USB chip type into RT2870_USB_DEVICES.

```
#define RT2870_USB_DEVICES  \
{   \
    {USB_DEVICE(0x148F,0x2770)}, /* Ralink */       \
    {USB_DEVICE(0x148F,0x2870)}, /* Ralink */       \
    {USB_DEVICE(0x148F,0x2070)}, /* Ralink 2070 */  \
    {USB_DEVICE(0x148F,0x3070)}, /* Ralink 3070 */  \
    .
    .
    .
}
```

e.g., VENDOR_ID = 0x148F, DEVICE_ID = 0x2770.

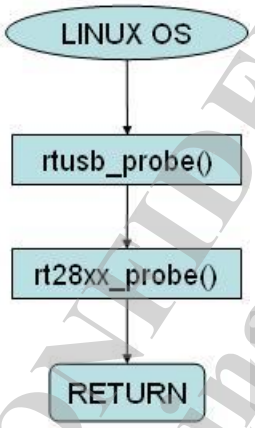


Figure 3-2 USB INIT flow chart

3.1.1.1.3 MAIN PROBE FUNCTION RT28XX_PROBE()

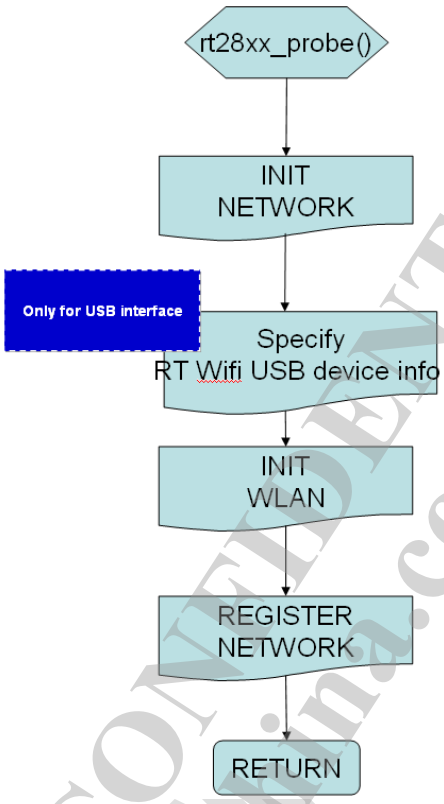


Figure 3-3 USB main probe function flow chart

Details

1. INIT NETWORK

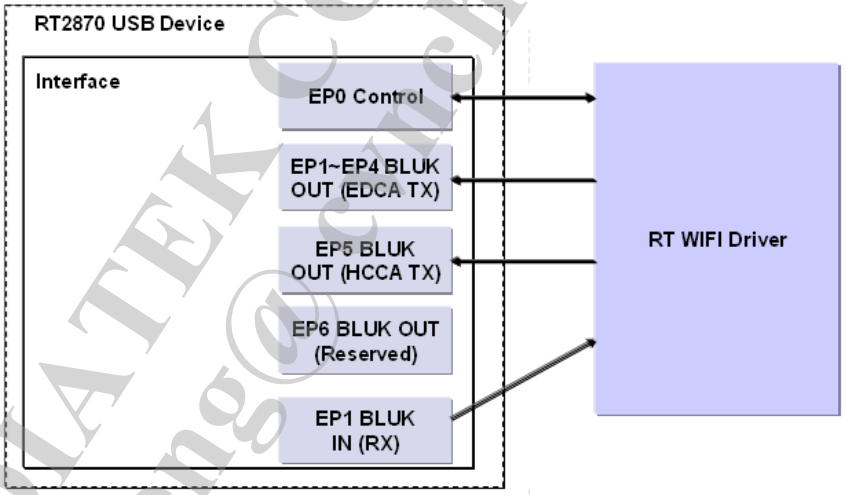
    a. First, allocate a network device & initialize it.

    b. RTMPAllocAdapterBlock(): Allocate driver control block & spin locks. Assign the control block pointer to the private data of the network device (→priv).

    c. USBDevConfigInit(): For RT Wi-Fi device base on USB interface, we will get additional USB characteristic sent by USB core such as endpoint address and maximum packet size. For example, below diagram is RT2870 endpoint layout

Endpoint	Function	Direction	Size	Tx Priority
EP0	Control I/O	IN/OUT	64	N/A
EP1OUT	EDCA Tx	OUT	512x2	Depend on WMM scheduling
EP2OUT	EDCA Tx	OUT	512x2	Depend on WMM scheduling
EP3OUT	EDCA Tx	OUT	512x2	Depend on WMM scheduling
EP4OUT	EDCA Tx	OUT	512x2	Depend on WMM scheduling
EP5OUT	HCCA Tx	OUT	512	Second
EP6OUT	Reserved	OUT	512	Highest
EP1IN	Rx	IN	512x2	N/A

RT2870 USB endpoints are EP0 control endpoints for command transfer, such as mac, bbp register, EEPROM, flash read/write control, EP1~EP4 bulk out endpoint for EDCA TX, EP5 bulk out endpoint for HCCA TX, and EP1 bulk in endpoint for RX . The maximum packet sizes of these bulk type USB transfers is 512 bytes. The endpoint descriptor is from the USB interface descriptor. In the future, the RT Wi-Fi driver can based on the bulk out (TX), bulk in (RX) transaction information, as shown in the subsequent figure.



d. RT28XXNetDevInit(): Initialize the network device (e.g., base address, irq number, etc.)

2. INIT WLAN

a. First, assign operation mode

b. For AP mode, pAd→OpMode = OPMODE_AP

c. For STA mode, pAd→OpMode = OPMODE_STA

d. Moule initialization

3. Multiple-Card & WSC

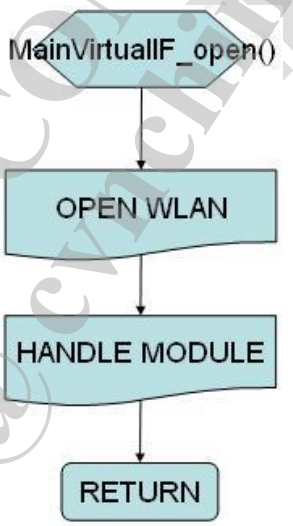
a. REGISTER NETWORK

i. rt_ieee80211_if_setup(): assign network operations & network interface name.

ii. Register the network interface to the LINUX.

3.1.1.2 Device Open

STA



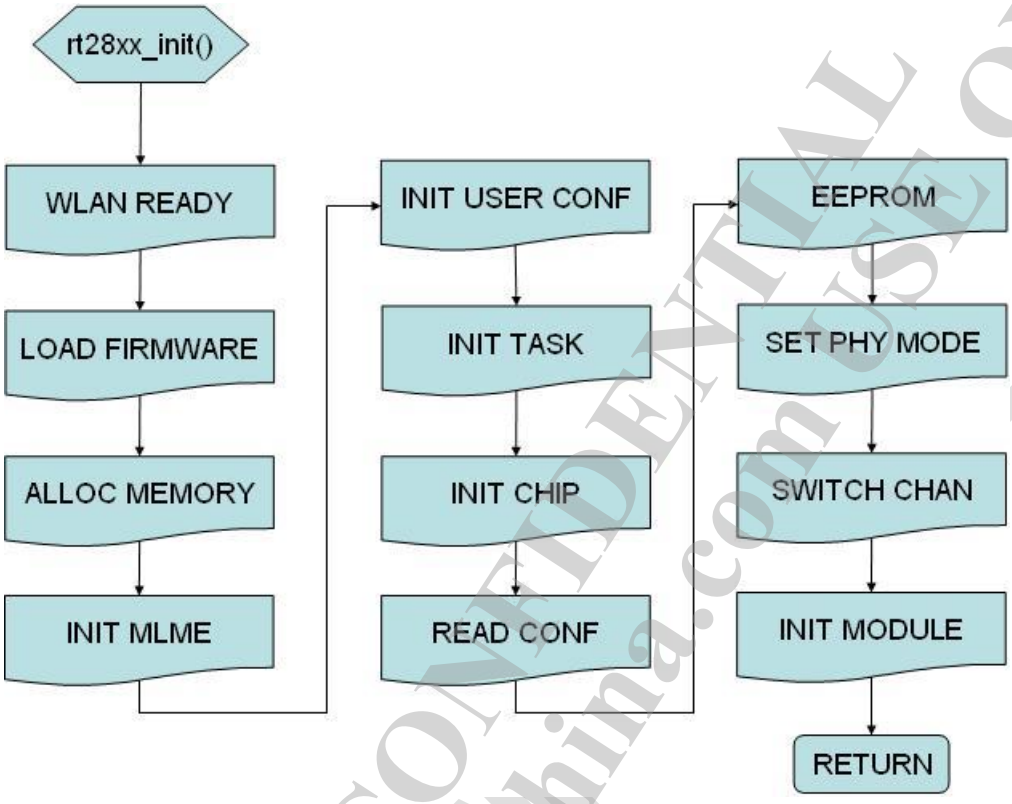


Figure 3-4 STA device open flow chart

Details

1. OPEN WLAN

    a. Register the IRQ handler to the LINUX.

    b. rt28xx_init(): WLAN Driver Initialization

- WLAN READY - Waiting for MAC ready by reading MAC_CSR0. Disable TX/RX DMA function in WPDMA_GLO_CFG.
- LOAD FIRMWARE - Load firmware to the internal 8051 and reset MCU hardware.
- ALLOC MEMORY - Allocate TX/RX buffer descriptors and RX packet buffer.
- INIT MLME - Initialize MLME queue, authentication/association FSM, various module FSM, periodical timer.
- INIT USER CONF - Initialize default user configuration.
- INIT TASK - Initialize tasks.
- INIT CHIP - Initialize default MAC/BBP registers.
- READ CONF - Read/Parse the configuration file.

- EEPROM - Read MAC/BBP register setting from EEPROM, FLASH, or Efuse.
- Set the settings to MAC/BBP registers.
- SET PHY MODE - Initialize the PHY mode to bg, a-only, bgn, etc.
- Initialize the channel list, default support/extended rate list.
- SWITCH CHAN - Switch current channel based on different BBP registers.
- INIT MODULE - Various Module initialization.

c. Various Module initialization.

d. Enable WLAN IRQ.

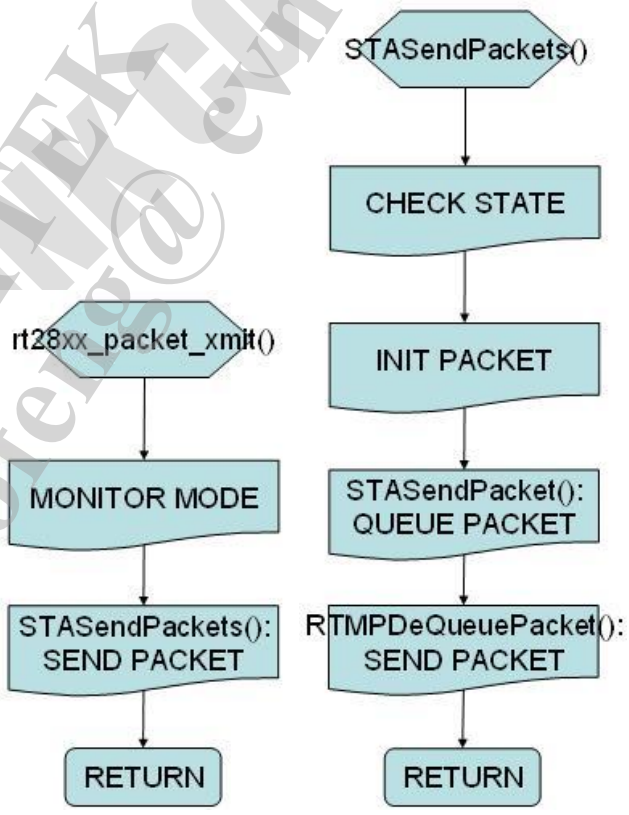
2. HANDLE MODULE

a. Increase module use count by 1

3.2 TRANSMISSION BLOCK

3.2.1 STA

1. SEND PACKET



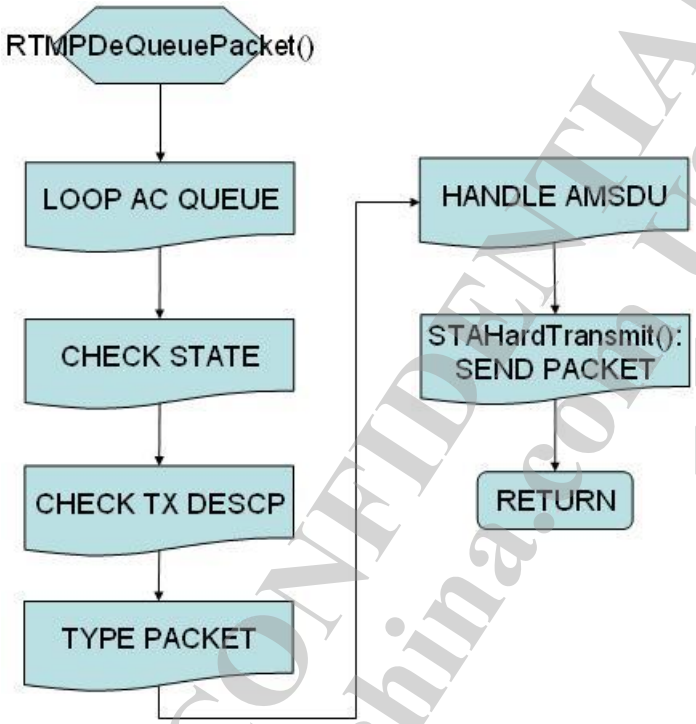


Figure 3-5 STA packet send flow chart

2. HANDLE PACKET TX COMPLETION

   a. PCI

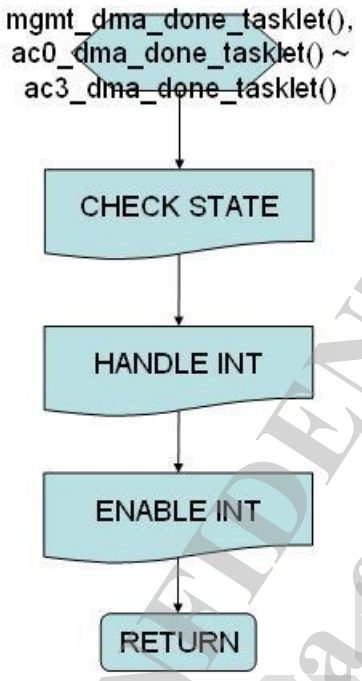


Figure 3-6 STA packet send completion flow chart (PCI)

mgmt_dma_done_tasklet() for any management frame.

ac0_dma_done_tasklet() for any data frame in AC0 queue.

ac1_dma_done_tasklet() for any data frame in AC1 queue.

ac2_dma_done_tasklet() for any data frame in AC2 queue.

ac3_dma_done_tasklet() for any data frame in AC3 queue.

- CHECK STATE - Check if current station driver state is on HALT/NIC NOT EXIST; if yes, do nothing.
- HANDLE INT - Mark the flag to indicate the interrupt will be handled, not pending state. Release the transmitted packet buffer in the ISR.
- ENABLE INT - Check if the pending flag is set; if yes, re-start the tasklet to handle the interrupt; if no, enable the interrupt again.

3. USB

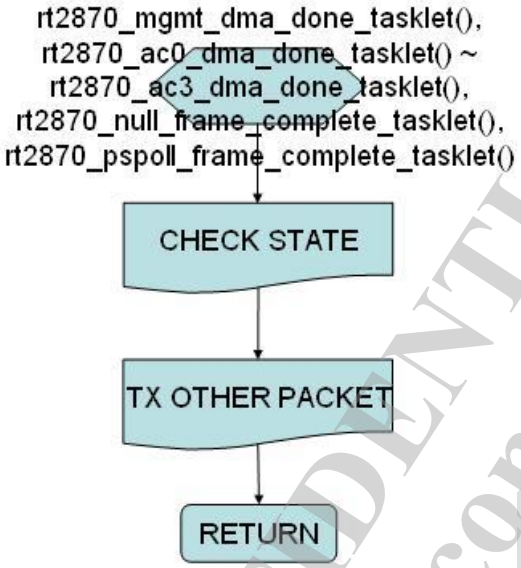


Figure 3-7 STA packet send completion flow chart (USB)

RTUSBBulkOutMLMEPacketComplete() for any management frame. ➔ will call rt2870_mgmt_dma_done_tasklet()

RTUSBBulkOutDataPacketComplete() for any data frame in AC0 ~ AC3 queue. ➔ will call rt2870_ac0_dma_done_tasklet() ~ rt2870_ac3_dma_done_tasklet()

RTUSBBulkOutNullFrameComplete() for any NULL frame. ➔ will call rt2870_null_frame_complete_tasklet()

RTUSBBulkOutPsPollComplete() for any PS-Poll frame ➔ will call rt2870_pspoll_frame_complete_tasklet()

- CHECK STATE - Check if current station driver state is on RESETING/HALT/NIC NOT EXIST; if yes, do nothing.
- TX OTHER PACKET - Check if any bulk out reset event occurs; if yes, send a command to USB COMMAND THREAD to do hardware reset. Check if any pending packet is queued in the software queue, try to send it.

Details

1. rt28xx_packet_xmit()

a. MONITOR MODE - Check if current station driver status is on MONITOR mode; if yes, drop the packet.

b. STASendPackets(): SEND PACKET

2. STASendPackets()

a. CHECK STATE - Check if current station driver state is on RESETING/HALT/RADIO OFF; if yes, drop the packet.

b. INIT PACKET - Initialize the private data of the packet, i.e. cb[] of LINUX SKB packet structure. e.g., packet source, packet status, etc.

c. STASendPacket(): QUEUE PACKET - Queue the packet into the software queue of the AC. (AC0 ~ AC3)

d. RTMPDeQueuePacket(): SEND PACKET - Write the packet to WLAN MAC and wait for transmitting.

3. RTMPDeQueuePacket()

a. LOOP AC QUEUE - We will get packets from the software queue of AC0 to AC3.

b. If the number of packets sent is larger than assigned maximum packet number, exit the routine and other queued packets will be transmitted at next time.

c. CHECK STATE - Check if current station driver state is on BSS SCANNING/RESETING/HALT/RADIO OFF/NIC NOT EXIST; if yes, do nothing.

d. CHECK TX DESCP - Early check to make sure we have enoguh TX Buffer Descriptor.

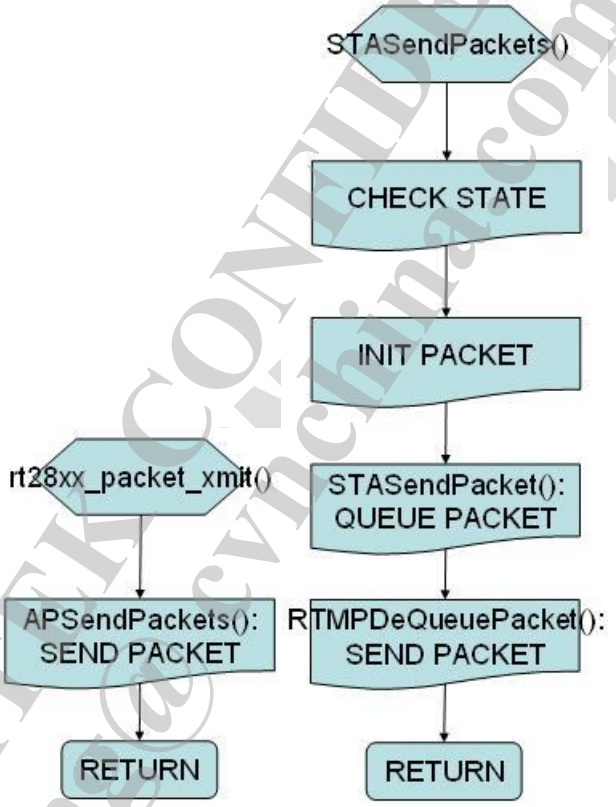
e. TYPE PACKET - Partition the packet to MULTICAST, LEGACY, AMPDU, AMSDU, RALINK PROPRIETARY AGGREGATION, or FRAGMENT type.

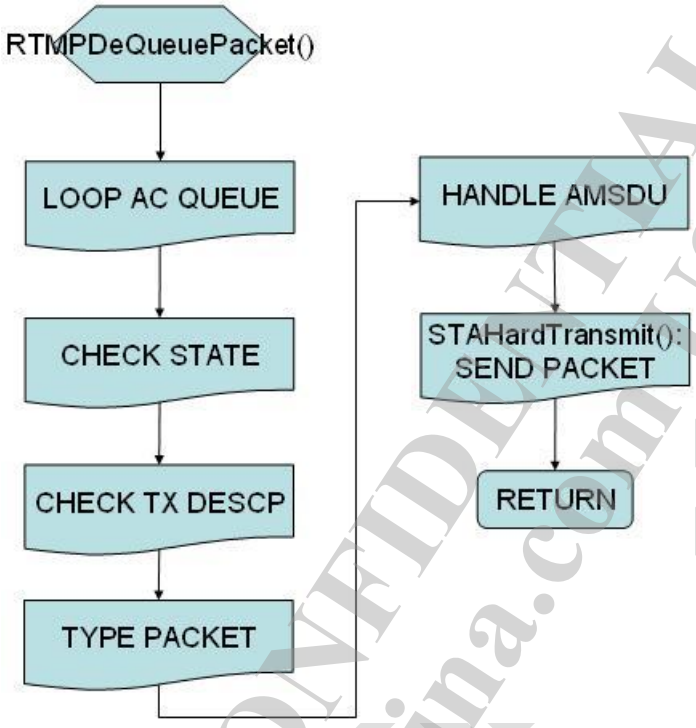
f. HANDLE AMSDU - Check if more than two packets can be aggregated for RALINK PROPRIETARY AGGREGATION or AMSDU type.

g. STAHardTransmit(): SEND PACKET - Build the IEEE802.11 WLAN header. Based on different packet type, such as AMPDU, AMSDU, etc. initialize the buffer descriptor, such as the packet pointer, packet length, etc. Finally, kick out the ASIC to send the packet.

3.2.2 AP

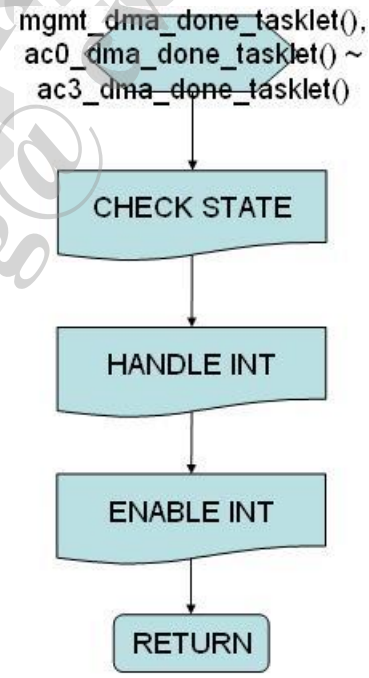
1. SEND PACKET





2. HANDLE PACKET TX COMPLETION

a. PCI



mgmt_dma_done_tasklet() for any management frame.

ac0_dma_done_tasklet() for any data frame in AC0 queue.

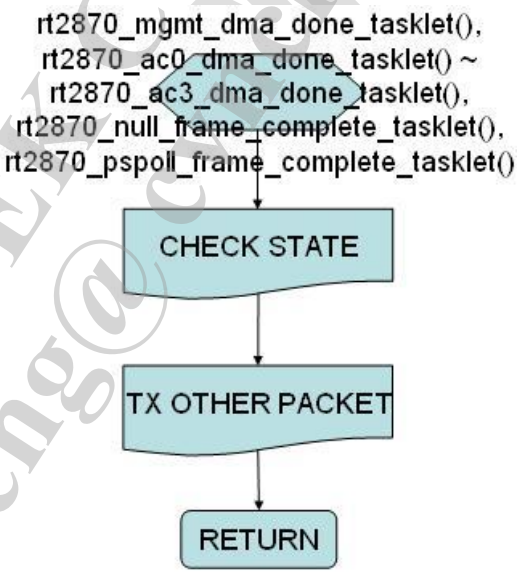
ac1_dma_done_tasklet() for any data frame in AC1 queue.

ac2_dma_done_tasklet() for any data frame in AC2 queue.

ac3_dma_done_tasklet() for any data frame in AC3 queue.

- CHECK STATE - Check if current station driver state is on HALT/NIC NOT EXIST; if yes, do nothing.
- HANDLE INT - Mark the flag to indicate the interrupt will be handled, not pending state. Release the transmitted packet buffer in the ISR.
- ENABLE INT - Check if the pending flag is set; if yes, re-start the tasklet to handle the interrupt; if no, enable the interrupt again.

4. USB



RTUSBBulkOutMLMEPacketComplete() for any management frame. ➔ will call rt2870_mgmt_dma_done_tasklet()

RTUSBBulkOutDataPacketComplete() for any data frame in AC0 ~ AC3 queue. ➔ will call rt2870_ac0_dma_done_tasklet() ~ rt2870_ac3_dma_done_tasklet()

RTUSBBulkOutNullFrameComplete() for any NULL frame. ➔ will call rt2870_null_frame_complete_tasklet()

RTUSBBulkOutPsPollComplete() for any PS-Poll frame ➔ will call rt2870_pspoll_frame_complete_tasklet()

- CHECK STATE - Check if current station driver state is on RESETING/HALT/NIC NOT EXIST; if yes, do nothing.
- TX OTHER PACKET - Check if any bulk out reset event occurs; if yes, send a command to USB COMMAND THREAD to do hardware reset. Check if any pending packet is queued in the software queue, try to send it.

Details

4. rt28xx_packet_xmit()
    a. MONITOR MODE - Check if current station driver status is on MONITOR mode; if yes, drop the packet.
    b. STASendPackets(): SEND PACKET
5. STASendPackets()
    a. CHECK STATE - Check if current station driver state is on RESETING/HALT/RADIO OFF; if yes, drop the packet.
    b. INIT PACKET - Initialize the private data of the packet, i.e. cb[] of LINUX SKB packet structure. e.g., packet source, packet status, etc.
    c. STASendPacket(): QUEUE PACKET - Queue the packet into the software queue of the AC. (AC0 ~ AC3)
    d. RTMPDeQueuePacket(): SEND PACKET - Write the packet to WLAN MAC and wait for transmitting.
6. RTMPDeQueuePacket()

a. LOOP AC QUEUE - We will get packets from the software queue of AC0 to AC3.

b. If the number of packets sent is larger than assigned maximum packet number, exit the routine and other queued packets will be transmitted at next time.

c. CHECK STATE - Check if current station driver state is on BSS SCANNING/RESETING/HALT/RADIO OFF/NIC NOT EXIST; if yes, do nothing.

d. CHECK TX DESCP - Early check to make sure we have enoguh TX Buffer Descriptor.

e. TYPE PACKET - Partition the packet to MULTICAST, LEGACY, AMPDU, AMSDU, RALINK PROPRIETARY AGGREGATION, or FRAGMENT type.

f. HANDLE AMSDU - Check if more than two packets can be aggregated for RALINK PROPRIETARY AGGREGATION or AMSDU type.

g. STAHardTransmit(): SEND PACKET - Build the IEEE802.11 WLAN header. Based on different packet type, such as AMPDU, AMSDU, etc. initialize the buffer descriptor, such as the packet pointer, packet length, etc. Finally, kick out the ASIC to send the packet.

3.3 RECEIVE BLOCK

3.3.1 STA

PCI

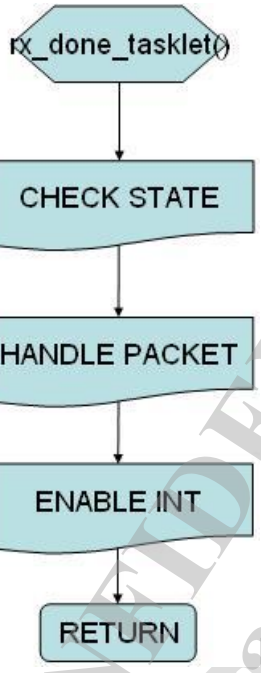


Figure 3-8 STA Receive block flowchart (PCI)

USB

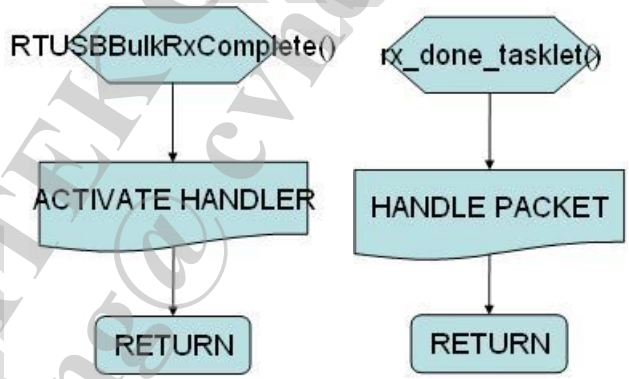
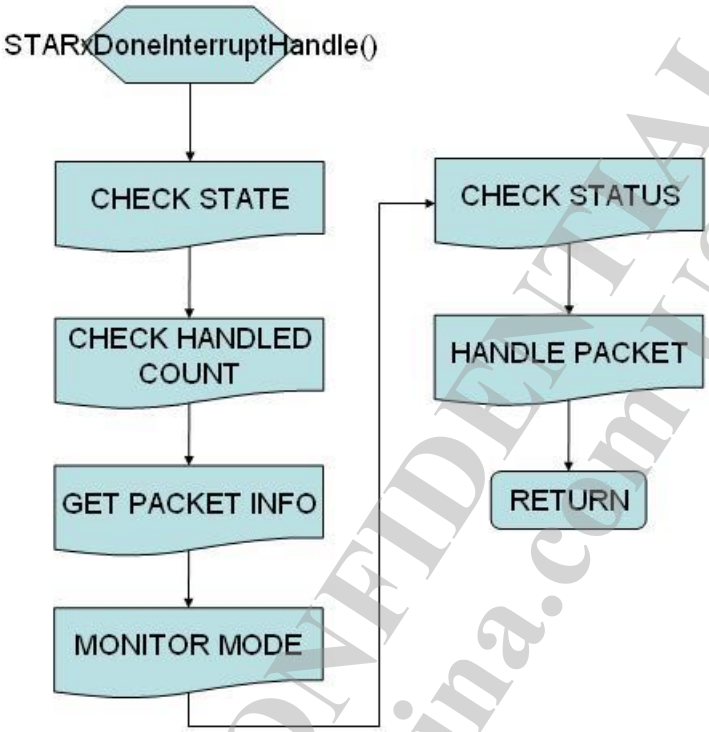
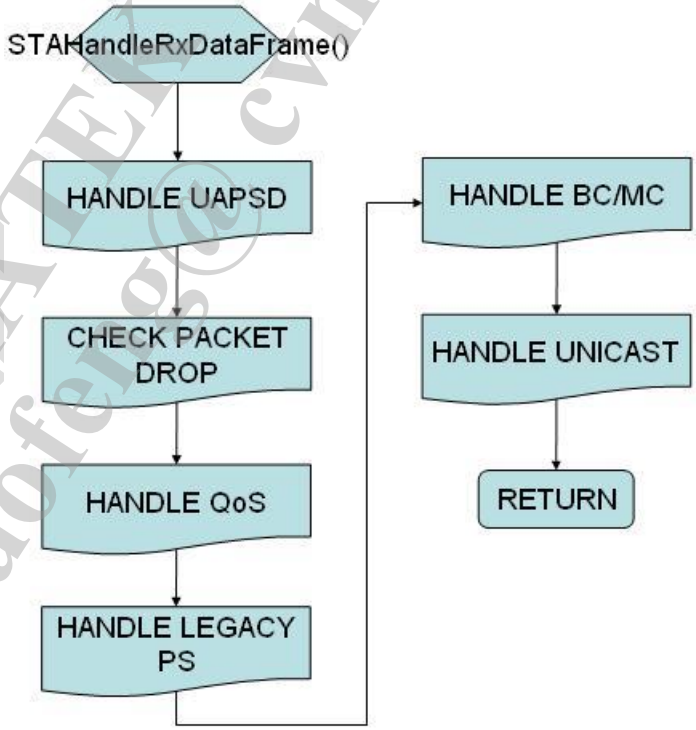


Figure 3-9 STA Receive block flowchart (USB)

Common





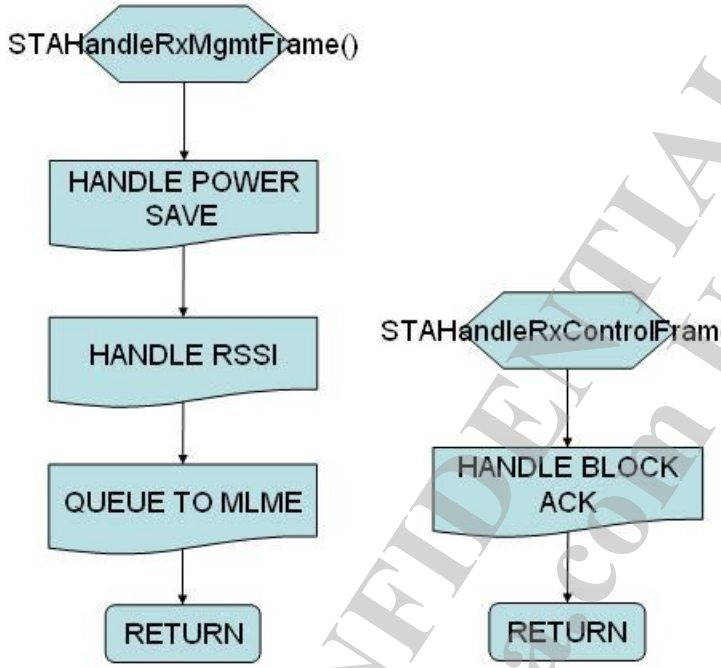


Figure 3-10 STA Receive block flowchart (PCI & USB)

Details

1. PCI

   a. rx_done_tasklet()

   - CHECK STATE - Check if current station driver state is on HALT/NIC NOT EXIST; if yes, do nothing.
   - HANDLE PACKET - Mark the flag to indicate the interrupt will be handled, not pending state. Handle any received management, data, control frame.
   - ENABLE INT - Check if the pending flag is set; if yes, re-start the tasklet to handle the interrupt; if no, enable the interrupt again.

2. USB

   a. RTUSBBulkRxComplete()

   - ACTIVATE HANDLER - Activate RX handler tasklet.

   b. rx_done_tasklet()

   - HANDLE PACKET - Increase the bulk in index to point the received packet.

- RTUSBBulkReceive(): Handle any received management, data, control frame. And submit a new receive URB to receive the next packet.

3. Common

a. STARxDoneInterruptHandle()

- CHECK STATE - Check if current station driver state is on RESETING/HALT/RADIO OFF/NIC NOT EXIST; if yes, do nothing.
- CHECK HANDLED COUNT - Check if the number of handled received packet is larger than the limitation; if yes, exit the routine and set the flag to re-activate the tasklet to handle other queued packets in hardware.
- GET PACKET INFO - Check if the Done bit of current received packet is set; if no, exit the routine and set the flag to re-activate the tasklet to handle other queued packets in hardware. Allocate a new receive packet buffer and assign it to current receive buffer descriptor. Activate the current receive buffer to receive new packet.
- MONITOR MODE - Check if current station driver status is on MONITOR mode; if yes, initialize the packet to a monitor packet and send it to the upper layer.
- CHECK STATUS - Check RSSI for Noise Hist statistic collection. Check ToDS bit of the packet header; if set, drop the packet. Check if the packet cannot be decrypted; if yes, trigger a event and drop the packet.
- HANDLE PACKET - Handle the packet based on management, control, or data frame.

b. STAHandleRxDataFrame()

- HANDLE UAPSD - If UAPSD function is enabled & the EOSP bit of received packet is set, force our chip to fall into sleep mode.
- CHECK PACKET DROP - If the received packet is NULL, CF-ACK(no data), CF-POLL(no data), or CF-ACK+CF-POLL(no data), drop it.
- HANDLE QoS - Get QoS Priority from the received packet header.
- HANDLE LEGACY PS - If the MoreData bit of received packet is set, send a PS-Poll control frame to the associated AP.
- HANDLE BC/MC - If the More-Fragment bit is set or the source address is us, drop the packet; Or forward the packet to the upper layer.
- HANDLE UNICAST - Update the RSSI statistics of received packet. If the More-Fragment bit is set, de-fragment the packet by calling RTMPDeFragmentDataFrame(). For encrypted fragment packet, calculate and check

the MIC of re-assembled packet. Forward the received packet to AMPDU process, AMSDU process, or the upper layer.

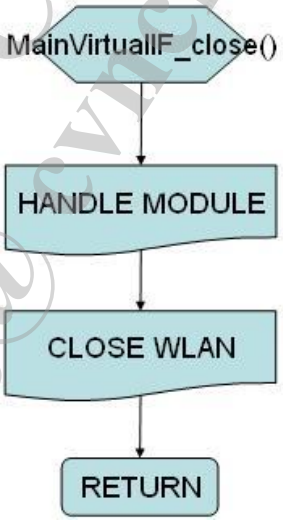
c. STAHandleRxMgmtFrame()

- HANDLE POWER SAVE - If we are in the PS mode and the MoreData bit of received management packet is set and the UAPSD state of VO queue (AC3) is not enabled, send a PS-Poll control frame to the assocaited AP.
- HANDLE RSSI - If the management packet is a beacon frame sent from the associated AP, update the RSSI statistics of received packet.
- QUEUE TO MLME - Queue the management packet to the queue of registered FSMs, such as AUTH, ASSOC, BLOCK ACK, etc.

d. STAHandleRxControlFrame()

- HANDLE BLOCK ACK - Handle the BLOCK ACK request control frame.

3.4 STOP BLOCK



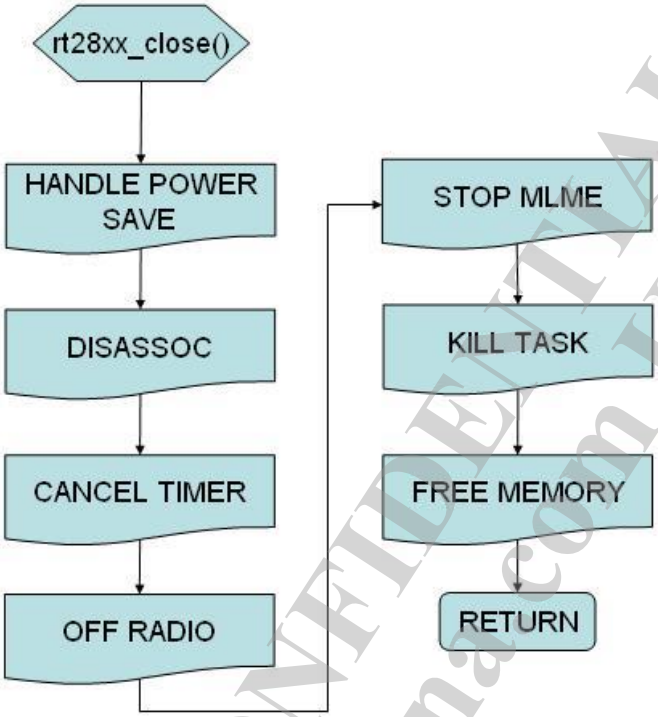


Figure 3-11 STA Stop block flowchart

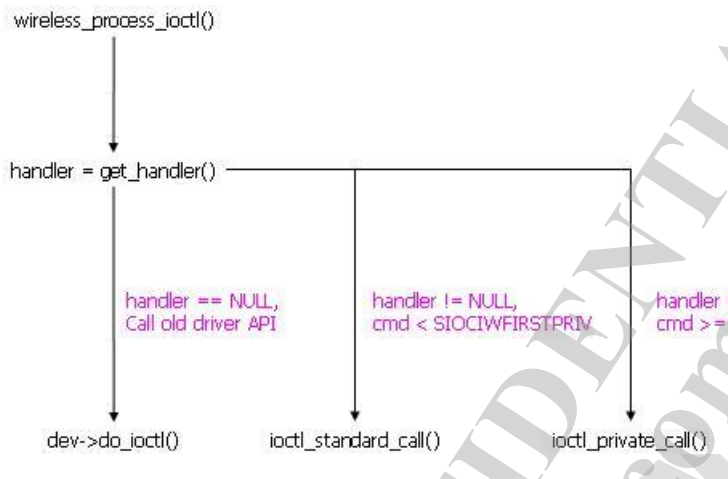
Details

1. HANDLE MODULE - Decrease module use count by 1.

    a. CLOSE WLAN - (a) rt28xx_close(): WLAN Driver Initialization

    - HANDLE POWER SAVE - Force to wake up chipset.
    - DISASSOC - Send a dis-association management frame to the associated AP.
    - CANCEL TIMER - Cancel used timers, such as rate switch timer, etc.
    - OFF RADIO - Disable related registers, LED, etc.
    - STOP MLME - Stop Mlme state machine, i.e. cancel any FSM timer.
    - KILL TASK - Kill any activated task.
    - FREE MEMORY - Free any allocated memory, such as control block, packet buffer, packet buffer descriptor, etc.

3.5 IOCTL BLOCK



In LINUX, we only have 32 private IOCTL command ID as below:

```
/* These 32 ioctl are wireless device private, for 16 commands.
 * Each driver is free to use them for whatever purpose it chooses,
 * however the driver *must* export the description of those ioctls
 * with SIOCGIWPRIV and *must* use arguments as defined below.
 * If you don't follow those rules, DaveM is going to hate you (reason :
 * it make mixed 32/64bit operation impossible).
 */
#define SIOCIWFIRSTPRIV 0x8BE0
#define SIOCIWLASTPRIV  0x8BFF
```

You can find RALINK PRIVATE COMMAND ID in include/old.h, from RT_PRIV_IOCTL to RTPRIV_IOCTL_SHOW.

Wireless extension versions released after version 12 support the specific Wireless Extension API (used in iwconfig and iwpriv utility) as shown below. The old driver API, dev→do_ioctl() in wireless_process_ioctl() will not be passed.

```
#if WIRELESS_EXT >= 12
    if (pAd->OpMode == OPMODE_STA)
    {
        dev->wireless_handlers = &rt28xx_iw_handler_def;
    }
#endif //WIRELESS_EXT >= 12
```

```
const struct iw_handler_def rt28xx_iw_handler_def =
{
#define N(a)    (sizeof (a) / sizeof (a[0]))
    .standard    = (iw_handler *) rt_handler,
    .num_standard    = sizeof(rt_handler) / sizeof(iw_handler),
    .private    = (iw_handler *) rt_priv_handlers,
    .num_private        = N(rt_priv_handlers),
    .private_args    = (struct iw_priv_args *) privtab,
    .num_private_args    = N(privtab),
#if IW_HANDLER_VERSION >= 7
    .get_wireless_stats = rt28xx_get_wireless_stats,
#endif
};
```

For normal IOCTL from the socket layer, the flow will pass the dev→do_ioctl(), i.e. rt28xx_ioctl().

The reference notes have more information about the supported commands and their usage.

3.5.1 STA

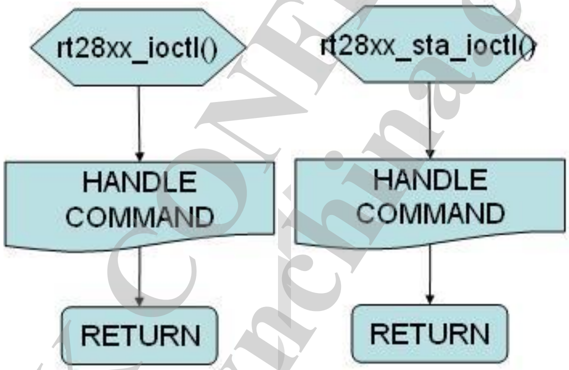


Figure 3-12 STA IOCTL block flowchart

Details

1. rt28xx_ioctl()
    a. HANDLE COMMAND - Call rt28xx_sta_ioctl() to handle the IOCTL.
2. rt28xx_sta_ioctl()
    a. HANDLE COMMAND - Handle various IOCTL from socket layer or iwpriv application.

3.6 USB BLOCK

3.6.1 TX Bulk OUT Aggregation

The RT Wi-Fi USB device uses “bulk aggregation” to enhance USB throughput. The RT Wi-Fi driver can aggregate multiple frame on the TX path into a “supper frame”, and pass it through the USB bus to RT Wi-Fi USB device. The on-chip USB DMA will automatically de-aggregate these frames according to their length, as specified in TXINO. The process is shown in the following figure.

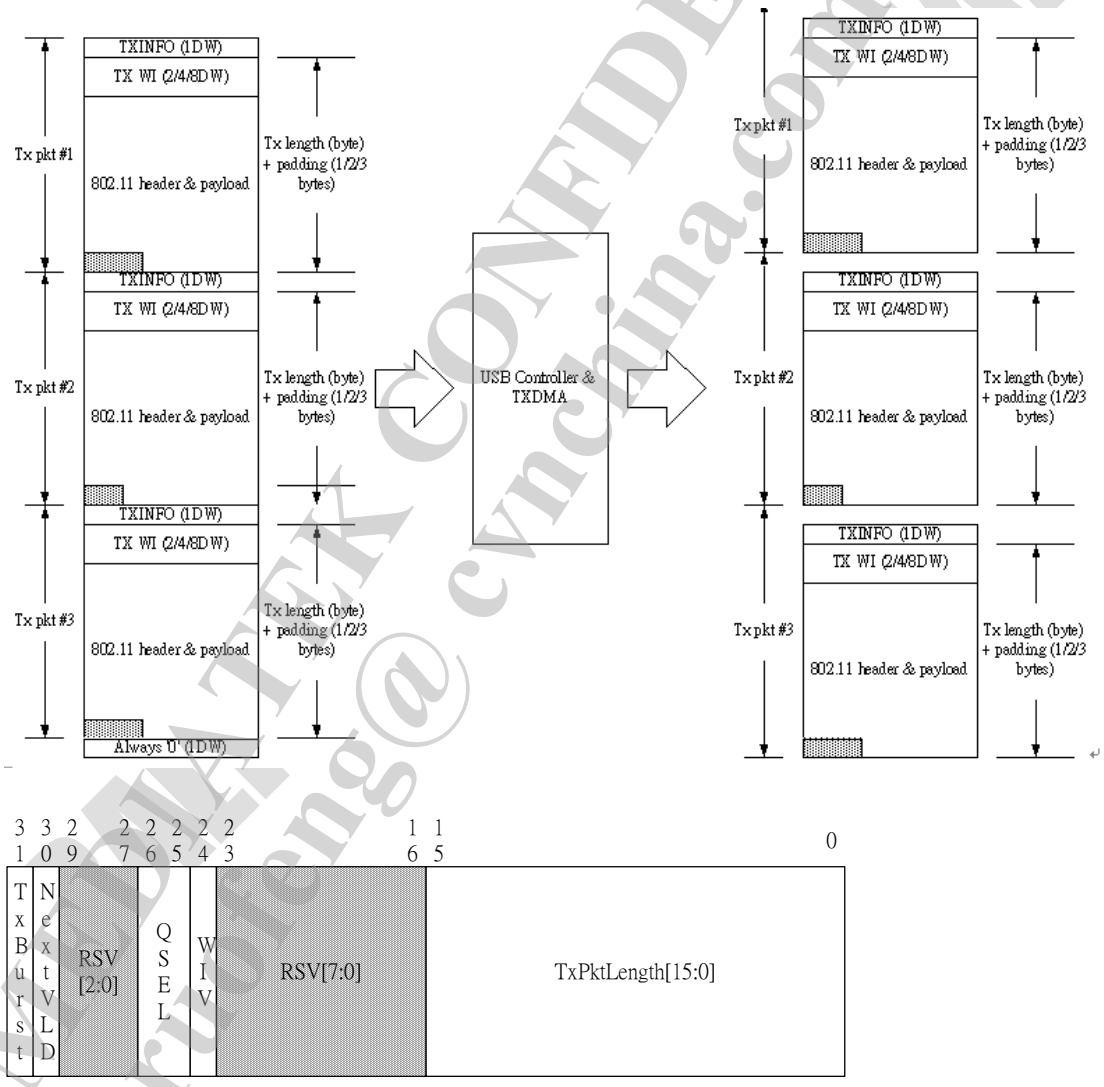


Figure 3-13 TXINFO structure

- TXBurst: Force USB DMA transmit frame from current selected endpoint

- NextVLD: Host driver info USB DMA current is not the last frame in current TX queue
- QSEL[1:0]:Packet buffer Q selection

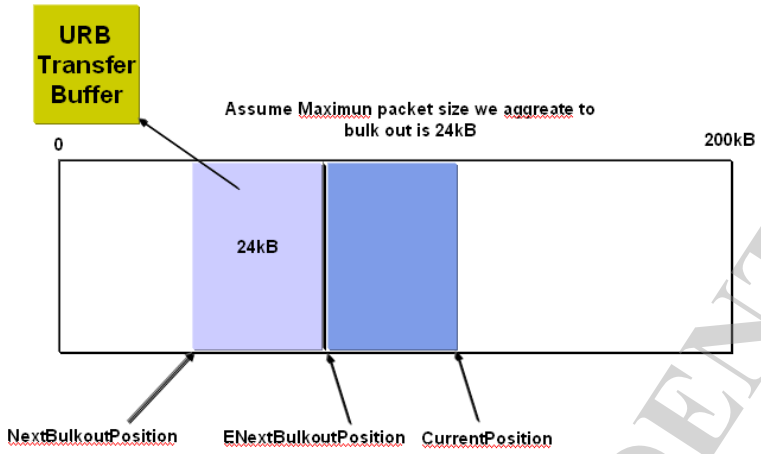
QSEL	Dest. In PBF	Function	Tx Priority
2'b00	Tx0Q	Management	Highest
2'b01	Tx1Q	HCCA	Medium
2'b10	Tx2Q	EDCA	Lowest
2'b11	N/A	N/A	N/A

- WIV: Wireless information valid (WI) valid
- TxPacketLength[15:0]: This field specifies the frame length in bytes. It includes WI, 802.11 header, and payload, but not TXINFO.

Because of the USB device memory buffer limitation, aggregated packet buffer sent to bulk out endpoint will be smaller than 32KB. The RT Wi-Fi driver has a 200KB TX buffer for each data type TX context transfer, and 3 parameters to control each bulk out. The parameters that control each bulk out are CurWritePosition, NextBulkOutPosition, and ENextBulkOutPosition.

- CurWritePosition - stores TX context buffer at the current write position
- NextBulkOutPosition - stores the next bulk out TX position
- ENextBulkOutPosition - stores the next bulk out position end

Below figure is pointer URB transfer to 24kB (assume 24KB byte is limitation) TX buffer they want to bulk out to USB device endpoint



The flow chart below shows that after sending, the USB host driver will call callback function RTUSBBulkOutDataPacketComplete to send the next bulk out data packet.

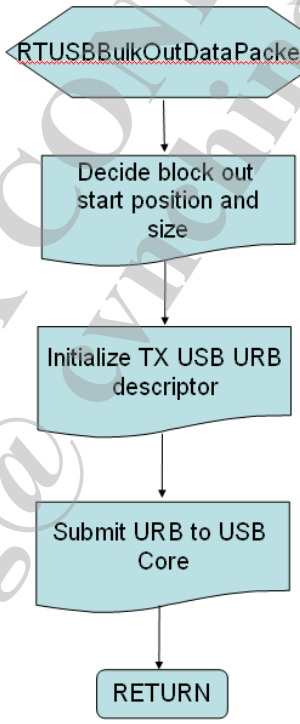
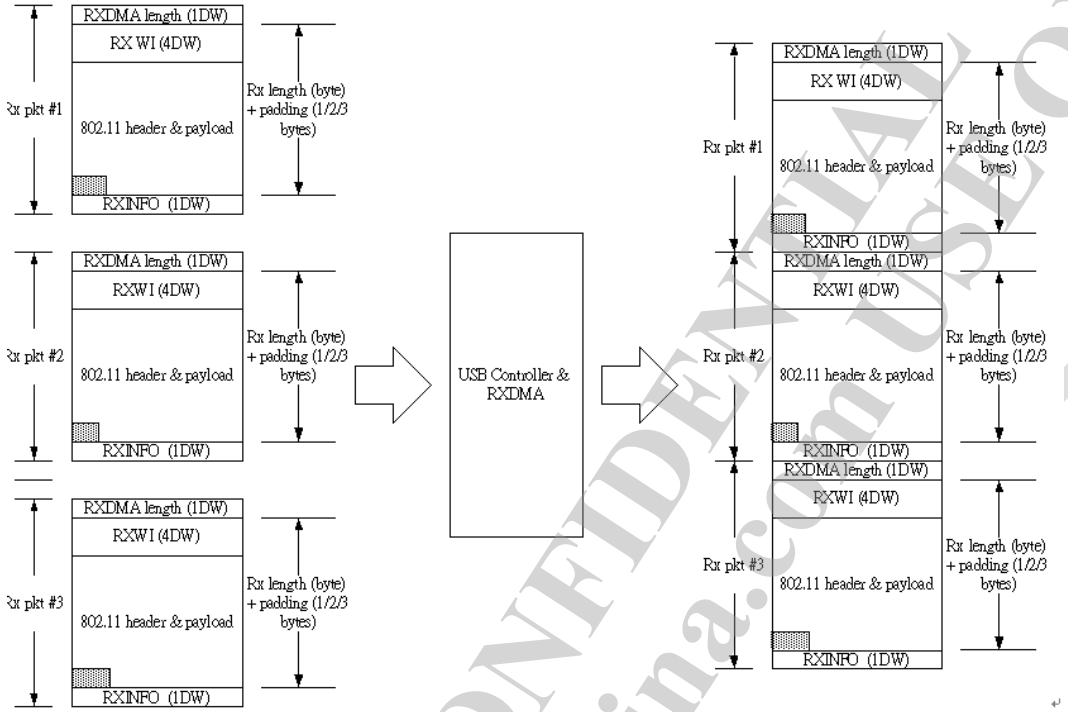


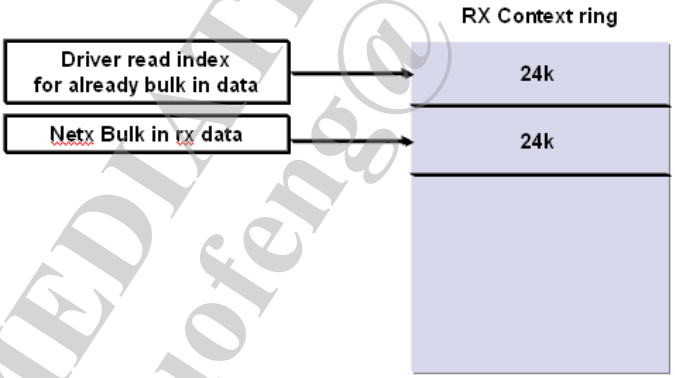
Figure 3-14 USB TX Bulk Out flow chart

3.6.2 RX Bulk IN Aggregation

On the RX path, USB DMA will aggregate continuous RX frames into a “super frame” to send to the USB bus. The maximum aggregated size can be set using the USB DMA register. The RT Wi-Fi driver recovers the original frames by reading the RXDMA length at the head of each RX frame



The driver is designed with an RX context ring to receive RX bulk IN data. Each context has 24kB and two fields: NextRXBulkInIndex, to do next RX URB bulk IN; and NextRxBulkInReadIndex, to read the RX "bulk in" data that has already been done in RX URB bulk IN transfer (received RX data from the USB device.)



DoBulkIn is called first. It gets the next bulk in RX Context element, and init RX URB descriptor, and submit URB to USB core. After the bulk in is completed, the length received is reduced from 24k to empty. The subsequent flow chart shows the jump to the next RX context element of the ring.

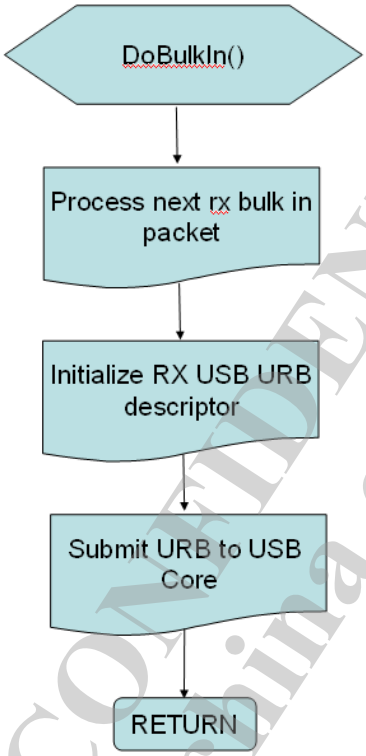


Figure 3-15 USB RX Bulk In flow chart A

When the bulk in transfer is completed, the USB host driver calls RTUSBBulkRxComplete. It creates a tasklet (rx_done_tasklet) to handle the RX frame, including analyzing which kind of 802.11 frame it is, and de-aggregating these frames if needed by reading the RXDMA length at the head of each RX frame mentioned above. It finally performs a bulk IN again. The following flow chart shows this procedure.

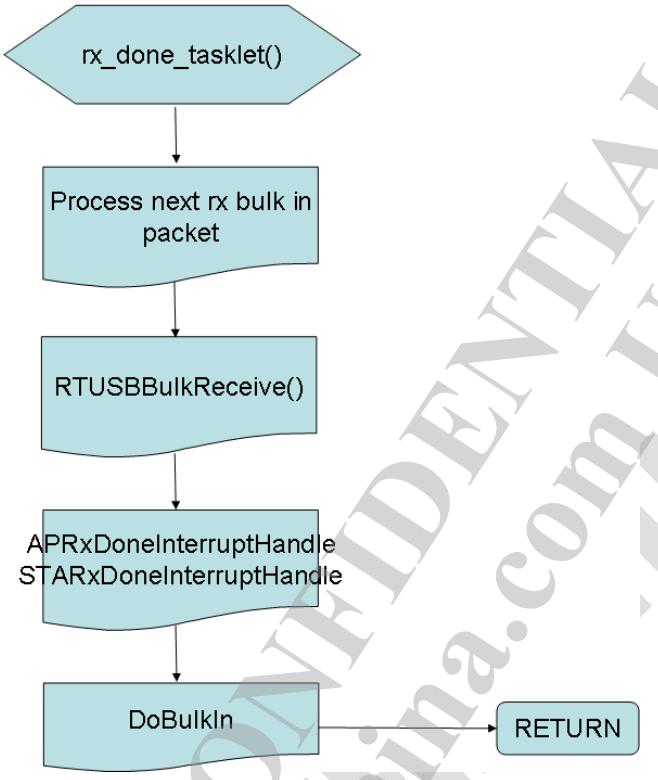


Figure 3-16 USB RX Bulk In flow chart B

3.7 OTHER BLOCK

3.7.1 INTERRUPT (only in PCI)

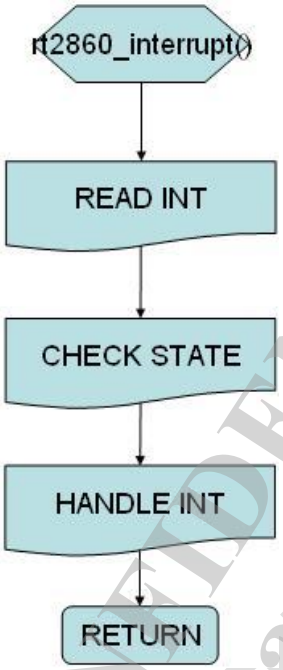


Figure 3-17 PCI Interrupt flow chart

Details

1. READ INT - Backup interrupt status and clear them.
2. CHECK STATE - Check if current station driver state is on HALT/NIC NOT EXIST; if yes, do nothing.
3. HANDLE INT - Check if current interrupt is disabled; if not, disable the interrupt and call the handler to process the interrupt and mark a flag to indicate the interrupt is pending to process.

3.7.2 FSM Overview

1. STATE_MACHINE in the mlme.h

```
typedef struct _STATE_MACHINE {
    ULONG                       Base;
    ULONG                       NrState;
    ULONG                       NrMsg;
    ULONG                       CurrState;
    STATE_MACHINE_FUNC          *TransFunc;
} STATE_MACHINE, *PSTATE_MACHINE;
```

Base: handler base index, always 0

NrState: maximum number of states

NrMsg: maximum number of events in a state

CurrState: current state

*TransFunc: all handers in all state, maximum number = (NrState × NrMsg), size = sizeof(VOID *) × (NrState × NrMsg)

2. INIT - Use StateMachineInit() to do FSM Initialization. You must prepare your STATE_MACHINE and STATE_MACHINE_FUNC[], e.g., struct _MLME_STRUCT {} of rtmp.h

3. SET EVENT HANDLER- Use StateMachineSetAction() to set your handler for the event in the state.

3.7.3 STA FSM

3.7.3.1 SYNC FSM

The FSM is responsible for finding APs.

1. STATE_MACHINE (rtmp.h and rtmp_def.h) - Declare SyncMachine and SyncFunc[SYNC_FUNC_SIZE].

```
#define SYNC_IDLE                       0
#define JOIN_WAIT_BEACON                1
#define SCAN_LISTEN                     2
#define MAX_SYNC_STATE                  3

#define SYNC_MACHINE_BASE               0
#define MT2_MLME_SCAN_REQ               0
#define MT2_MLME_JOIN_REQ               1
#define MT2_MLME_START_REQ              2
#define MT2_PEER_BEACON                 3
#define MT2_PEER_PROBE_RSP              4
#define MT2_PEER_ATIM                   5
#define MT2_SCAN_TIMEOUT                6
#define MT2_BEACON_TIMEOUT              7
#define MT2_ATIM_TIMEOUT                8
#define MT2_PEER_PROBE_REQ              9
#define MAX_SYNC_MSG                    10

#define SYNC_FUNC_SIZE                  (MAX_SYNC_STATE * MAX_SYNC_MSG)
```

2. INIT & SET EVENT HANDLER (sync.c and mlme.c)

```
VOID SyncStateMachineInit(
    IN PRTMP_ADAPTER pAd,
    IN STATE_MACHINE *Sm,
    OUT STATE_MACHINE_FUNC Trans[])
{
    StateMachineInit(Sm, Trans, MAX_SYNC_STATE, MAX_SYNC_MSG,
                     (STATE_MACHINE_FUNC)Drop, SYNC_IDLE, SYNC_MACHINE_BASE);

    // column 1
    StateMachineSetAction(Sm, SYNC_IDLE, MT2_MLME_SCAN_REQ,
                     (STATE_MACHINE_FUNC)MlmeScanReqAction);
    StateMachineSetAction(Sm, SYNC_IDLE, MT2_MLME_JOIN_REQ,
                     (STATE_MACHINE_FUNC)MlmeJoinReqAction);
    StateMachineSetAction(Sm, SYNC_IDLE, MT2_MLME_START_REQ,
                     (STATE_MACHINE_FUNC)MlmeStartReqAction);
    StateMachineSetAction(Sm, SYNC_IDLE, MT2_PEER_BEACON,
                     (STATE_MACHINE_FUNC)PeerBeacon);
    StateMachineSetAction(Sm, SYNC_IDLE, MT2_PEER_PROBE_REQ,
                     (STATE_MACHINE_FUNC)PeerProbeReqAction);

    //column 2
    StateMachineSetAction(Sm, JOIN_WAIT_BEACON, MT2_MLME_SCAN_REQ,
                     (STATE_MACHINE_FUNC)InvalidStateWhenScan);
    StateMachineSetAction(Sm, JOIN_WAIT_BEACON, MT2_MLME_JOIN_REQ,
                     (STATE_MACHINE_FUNC)InvalidStateWhenJoin);
    StateMachineSetAction(Sm, JOIN_WAIT_BEACON, MT2_MLME_START_REQ,
                     (STATE_MACHINE_FUNC)InvalidStateWhenStart);
    StateMachineSetAction(Sm, JOIN_WAIT_BEACON, MT2_PEER_BEACON,
                     (STATE_MACHINE_FUNC)PeerBeaconAtJoinAction);
    StateMachineSetAction(Sm, JOIN_WAIT_BEACON, MT2_BEACON_TIMEOUT,
                     (STATE_MACHINE_FUNC)BeaconTimeoutAtJoinAction);


    // column 3
    StateMachineSetAction(Sm, SCAN_LISTEN, MT2_MLME_SCAN_REQ,
                     (STATE_MACHINE_FUNC)InvalidStateWhenScan);
    StateMachineSetAction(Sm, SCAN_LISTEN, MT2_MLME_JOIN_REQ,
                     (STATE_MACHINE_FUNC)InvalidStateWhenJoin);
    StateMachineSetAction(Sm, SCAN_LISTEN, MT2_MLME_START_REQ,
                     (STATE_MACHINE_FUNC)InvalidStateWhenStart);
    StateMachineSetAction(Sm, SCAN_LISTEN, MT2_PEER_BEACON,
                     (STATE_MACHINE_FUNC)PeerBeaconAtScanAction);
    StateMachineSetAction(Sm, SCAN_LISTEN, MT2_PEER_PROBE_RSP,
                     (STATE_MACHINE_FUNC)PeerBeaconAtScanAction);
    StateMachineSetAction(Sm, SCAN_LISTEN, MT2_SCAN_TIMEOUT,
                     (STATE_MACHINE_FUNC)ScanTimeoutAction);

    // timer init
    RTMPInitTimer(pAd, &pAd->MlmeAux.BeaconTimer,
                  GET_TIMER_FUNCTION(BeaconTimeout), pAd, FALSE);
    RTMPInitTimer(pAd, &pAd->MlmeAux.ScanTimer,
                  GET_TIMER_FUNCTION(ScanTimeout), pAd, FALSE);
}
```

a. SYNC_IDLE state:

- Receive Event MT2_MLME_SCAN_REQ (IDLE-1) - Call MlmeScanReqAction () to do scan channel action.
- Receive Event MT2_MLME_JOIN_REQ (IDLE-2) - Call MlmeJoinReqAction
- Receive Event MT2_MLME_START_REQ (IDLE-3) - Call MlmeStartReqAction() to do Ad-hoc mode built.
- Receive Event MT2_PEER_BEACON (IDLE-4) - Call PeerBeacon() to handle the received beacon.

- Receive Event MT2_PEER_PROBE_REQ (IDLE-5) - Call PeerProbeReqAction() to handle the received probe request frame in the Ad-hoc mode.

b. JOIN_WAIT_BEACON state:

- Receive Event MT2_MLME_SCAN_REQ (JOIN-1)
- Call InvalidStateWhenScan() to reset SYNC FSM.
- Receive Event MT2_MLME_JOIN_REQ (JOIN-2)
- Call InvalidStateWhenJoin() to reset SYNC FSM.
- Receive Event MT2_MLME_START_REQ (JOIN-3)
- Call InvalidStateWhenStart() to reset SYNC FSM.
- Receive Event MT2_PEER_BEACON (JOIN-4)
- Call PeerBeaconAtJoinAction() to handle the received beacon.
- Receive Event MT2_BEACON_TIMEOUT (JOIN-5)
- Call BeaconTimeoutAtJoinAction() to reset SYNC FSM.

c. SCAN_LISTEN state:

- Receive Event MT2_MLME_SCAN_REQ (SCAN-1) - Call InvalidStateWhenScan() to reset SYNC FSM.
- Receive Event MT2_MLME_JOIN_REQ (SCAN-2) - Call InvalidStateWhenJoin() to reset SYNC FSM.
- Receive Event MT2_MLME_START_REQ (SCAN-3) - Call InvalidStateWhenStart() to reset SYNC FSM.
- Receive Event MT2_PEER_BEACON (SCAN-4) - Call PeerBeaconAtScanAction() to handle the received beacon.

- Receive Event MT2_PEER_PROBE_RSP (SCAN-5) - Call PeerBeaconAtScanAction() to handle the received probe response frame.
- Receive Event MT2_SCAN_TIMEOUT (SCAN-6) - Call ScanTimeoutAction() to scan next channel.

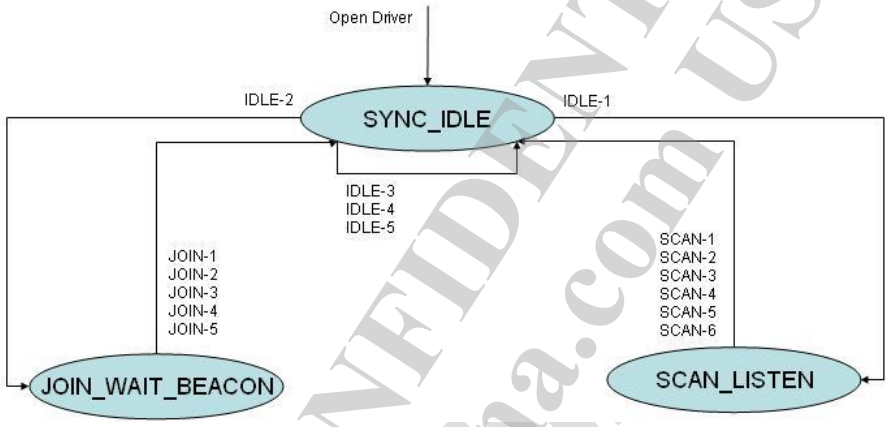


Figure 3-18 SYNC FSM flow chart

3.7.3.2 AUTH FSM

The Auth FSM is responsible for the authentication state built.

STATE_MACHINE (rtmp.h and rtmp_def.h)

Declare AuthMachine and AuthFunc[AUTH_FUNC_SIZE].

```
#define AUTH_REQ_IDLE                   0
#define AUTH_WAIT_SEQ2                  1
#define AUTH_WAIT_SEQ4                  2
#define MAX_AUTH_STATE                  3

#define AUTH_MACHINE_BASE               0
#define MT2_MLME_AUTH_REQ               0
#define MT2_PEER_AUTH_EVEN              1
#define MT2_AUTH_TIMEOUT                2
#define MAX_AUTH_MSG                    3

#define AUTH_FUNC_SIZE                  (MAX_AUTH_STATE * MAX_AUTH_MSG)
```

INIT & SET EVENT HANDLER (auth.c and mlme.c)

```
void AuthStateMachineInit(
    IN PRTMP_ADAPTER pAd,
    IN STATE_MACHINE *Sm,
    OUT STATE_MACHINE_FUNC Trans[])
{
    StateMachineInit(Sm, Trans, MAX_AUTH_STATE, MAX_AUTH_MSG,
                (STATE_MACHINE_FUNC)Drop, AUTH_REQ_IDLE, AUTH_MACHINE_BASE);

    // the first column
    StateMachineSetAction(Sm, AUTH_REQ_IDLE, MT2_MLME_AUTH_REQ,
                (STATE_MACHINE_FUNC)MlmeAuthReqAction);

    // the second column
    StateMachineSetAction(Sm, AUTH_WAIT_SEQ2, MT2_MLME_AUTH_REQ,
                (STATE_MACHINE_FUNC)InvalidStateWhenAuth);
    StateMachineSetAction(Sm, AUTH_WAIT_SEQ2, MT2_PEER_AUTH_EVEN,
                (STATE_MACHINE_FUNC)PeerAuthRspAtSeq2Action);
    StateMachineSetAction(Sm, AUTH_WAIT_SEQ2, MT2_AUTH_TIMEOUT,
                (STATE_MACHINE_FUNC)AuthTimeoutAction);

    // the third column
    StateMachineSetAction(Sm, AUTH_WAIT_SEQ4, MT2_MLME_AUTH_REQ,
                (STATE_MACHINE_FUNC)InvalidStateWhenAuth);
    StateMachineSetAction(Sm, AUTH_WAIT_SEQ4, MT2_PEER_AUTH_EVEN,
                (STATE_MACHINE_FUNC)PeerAuthRspAtSeq4Action);
    StateMachineSetAction(Sm, AUTH_WAIT_SEQ4, MT2_AUTH_TIMEOUT,
                (STATE_MACHINE_FUNC)AuthTimeoutAction);

    RTMPInitTimer(pAd, &pAd->MlmeAux.AuthTimer,
                GET_TIMER_FUNCTION(AuthTimeout), pAd, FALSE);
}
```

1. AUTH_REQ_IDLE state

- Receive Event MT2_MLME_AUTH_REQ. Call MlmeAuthReqAction() to send the authentication frame to the BSSID.

2. AUTH_WAIT_SEQ2 state

- Receive Event MT2_MLME_AUTH_REQ. Call InvalidStateWhenAuth() to reset AUTH FSM.
- Receive Event MT2_PEER_AUTH_EVEN. Call PeerAuthRspAtSeq2Action() to handle the authentication response frame.
- Receive Event MT2_AUTH_TIMEOUT. Call AuthTimeoutAction() to handle the authentication response timeout action.

3. AUTH_WAIT_SEQ4 state

- Receive Event MT2_MLME_AUTH_REQ. Call InvalidStateWhenAuth() to reset AUTH FSM.
- Receive Event MT2_PEER_AUTH_EVEN. Call PeerAuthRspAtSeq4Action() to handle the authentication ack frame.
- Receive Event MT2_AUTH_TIMEOUT. Call AuthTimeoutAction() to handle the authentication ack timeout action.

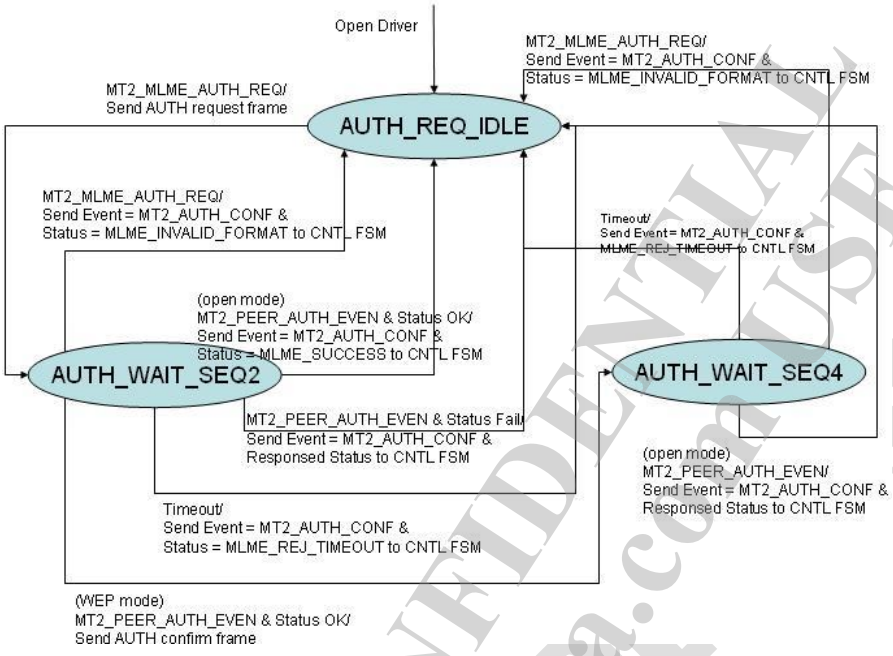


Figure 3-19 AUTH FSM flow chart

3.7.3.3 ASSOC FSM

The FSM is responsible for association state built.

STATE_MACHINE

Declare AssocMachine and AssocFunc[AP_ASSOC_FUNC_SIZE].

```
#define AP_ASSOC_IDLE                   0
#define AP_MAX_ASSOC_STATE              1

#define AP_ASSOC_MACHINE_BASE           0
#define APMT2_MLME_DISASSOC_REQ         0
#define APMT2_PEER_DISASSOC_REQ         1
#define APMT2_PEER_ASSOC_REQ            2
#define APMT2_PEER_REASSOC_REQ          3
#define APMT2_CLS3ERR                   4
#define AP_MAX_ASSOC_MSG                5

#define AP_ASSOC_FUNC_SIZE              (AP_MAX_ASSOC_STATE * AP_MAX_ASSOC_MSG)
```

1. INIT & SET EVENT HANDLER (assoc.c and mlme.c)

```
VOID AssocStateMachineInit(
    IN  PRTMP_ADAPTER   pAd,
    IN  STATE_MACHINE *S,
    OUT STATE_MACHINE_FUNC Trans[])
{
    StateMachineInit(S, Trans, MAX_ASSOC_STATE, MAX_ASSOC_MSG,
                     (STATE_MACHINE_FUNC)Drop, ASSOC_IDLE, ASSOC_MACHINE_BASE);

    // first column
    StateMachineSetAction(S, ASSOC_IDLE, MT2_MLME_ASSOC_REQ,
                     (STATE_MACHINE_FUNC)MlmeAssocReqAction);
    StateMachineSetAction(S, ASSOC_IDLE, MT2_MLME_REASSOC_REQ,
                     (STATE_MACHINE_FUNC)MlmeReassocReqAction);
    StateMachineSetAction(S, ASSOC_IDLE, MT2_MLME_DISASSOC_REQ,
                     (STATE_MACHINE_FUNC)MlmeDisassocReqAction);
    StateMachineSetAction(S, ASSOC_IDLE, MT2_PEER_DISASSOC_REQ,
                     (STATE_MACHINE_FUNC)PeerDisassocAction);

    // second column
    StateMachineSetAction(S, ASSOC_WAIT_RSP, MT2_MLME_ASSOC_REQ,
                     (STATE_MACHINE_FUNC)InvalidStateWhenAssoc);
    StateMachineSetAction(S, ASSOC_WAIT_RSP, MT2_MLME_REASSOC_REQ,
                     (STATE_MACHINE_FUNC)InvalidStateWhenReassoc);
    StateMachineSetAction(S, ASSOC_WAIT_RSP, MT2_MLME_DISASSOC_REQ,
                     (STATE_MACHINE_FUNC)InvalidStateWhenDisassociate);
    StateMachineSetAction(S, ASSOC_WAIT_RSP, MT2_PEER_DISASSOC_REQ,
                     (STATE_MACHINE_FUNC)PeerDisassocAction);
    StateMachineSetAction(S, ASSOC_WAIT_RSP, MT2_PEER_ASSOC_RSP,
                     (STATE_MACHINE_FUNC)PeerAssocRspAction);
```

```
    //
    // We send Assoc request frame to this AP,
    // it always send Reassoc Rsp not Associate Rsp.
    //
    StateMachineSetAction(S, ASSOC_WAIT_RSP, MT2_PEER_REASSOC_RSP,
                          (STATE_MACHINE_FUNC)PeerAssocRspAction);
    StateMachineSetAction(S, ASSOC_WAIT_RSP, MT2_ASSOC_TIMEOUT,
                          (STATE_MACHINE_FUNC)AssocTimeoutAction);

    // third column
    StateMachineSetAction(S, REASSOC_WAIT_RSP, MT2_MLME_ASSOC_REQ,
                          (STATE_MACHINE_FUNC)InvalidStateWhenAssoc);
    StateMachineSetAction(S, REASSOC_WAIT_RSP, MT2_MLME_REASSOC_REQ,
                          (STATE_MACHINE_FUNC)InvalidStateWhenReassoc);
    StateMachineSetAction(S, REASSOC_WAIT_RSP, MT2_MLME_DISASSOC_REQ,
                          (STATE_MACHINE_FUNC)InvalidStateWhenDisassociate);
    StateMachineSetAction(S, REASSOC_WAIT_RSP, MT2_PEER_DISASSOC_REQ,
                          (STATE_MACHINE_FUNC)PeerDisassocAction);
    StateMachineSetAction(S, REASSOC_WAIT_RSP, MT2_PEER_REASSOC_RSP,
                          (STATE_MACHINE_FUNC)PeerReassocRspAction);

    //
    // Patch, AP doesn't send Reassociate Rsp frame to Station.
    //
    StateMachineSetAction(S, REASSOC_WAIT_RSP, MT2_PEER_ASSOC_RSP,
                          (STATE_MACHINE_FUNC)PeerReassocRspAction);
    StateMachineSetAction(S, REASSOC_WAIT_RSP, MT2_REASSOC_TIMEOUT,
                          (STATE_MACHINE_FUNC)ReassocTimeoutAction);


    // fourth column
    StateMachineSetAction(S, DISASSOC_WAIT_RSP, MT2_MLME_ASSOC_REQ,
                          (STATE_MACHINE_FUNC)InvalidStateWhenAssoc);
    StateMachineSetAction(S, DISASSOC_WAIT_RSP, MT2_MLME_REASSOC_REQ,
                          (STATE_MACHINE_FUNC)InvalidStateWhenReassoc);
    StateMachineSetAction(S, DISASSOC_WAIT_RSP, MT2_MLME_DISASSOC_REQ,
                          (STATE_MACHINE_FUNC)InvalidStateWhenDisassociate);
    StateMachineSetAction(S, DISASSOC_WAIT_RSP, MT2_PEER_DISASSOC_REQ,
                          (STATE_MACHINE_FUNC)PeerDisassocAction);
    StateMachineSetAction(S, DISASSOC_WAIT_RSP, MT2_DISASSOC_TIMEOUT,
                          (STATE_MACHINE_FUNC)DisassocTimeoutAction);

    // initialize the timer
    RTMPInitTimer(pAd, &pAd->MlmeAux.AssocTimer,
                  GET_TIMER_FUNCTION(AssocTimeout), pAd, FALSE);
    RTMPInitTimer(pAd, &pAd->MlmeAux.ReassocTimer,
                  GET_TIMER_FUNCTION(ReassocTimeout), pAd, FALSE);
    RTMPInitTimer(pAd, &pAd->MlmeAux.DisassocTimer,
                  GET_TIMER_FUNCTION(DisassocTimeout), pAd, FALSE);
}
```

2. ASSOC_IDLE state

- Receive Event MT2_MLME_ASSOC_REQ (IDLE-1). Call MlmeAssocReqAction() to send a association request frame to the BSSID.
- Receive Event MT2_MLME_REASSOC_REQ (IDLE-2). Call MlmeReassocReqAction() to send a reassociation request frame to the BSSID.
- Receive Event MT2_MLME_DISASSOC_REQ (IDLE-3). Call MlmeDisassocReqAction() to send a dis-association request frame to the BSSID.
- Receive Event MT2_PEER_DISASSOC_REQ (IDLE-4). Call PeerDisassocAction() to handle the dis-association from the BSSID.

a. ASSOC_WAIT_RSP state:

- Receive Event MT2_MLME_ASSOC_REQ (ASSOC-1). Call InvalidStateWhenAssoc() to reset ASSOC FSM.
- Receive Event MT2_MLME_REASSOC_REQ (ASSOC-2). Call InvalidStateWhenReassoc() to reset REASSOC FSM.
- Receive Event MT2_MLME_DISASSOC_REQ (ASSOC-3). Call InvalidStateWhenDisassociate() to reset ASSOC FSM.
- Receive Event MT2_PEER_DISASSOC_REQ (ASSOC-4). Call PeerDisassocAction() to hand the dis-association frame from the BSSID.
- Receive Event MT2_PEER_ASSOC_RSP (ASSOC-5). Call PeerAssocRspAction() to handle the association response frame from the BSSID and send a event MT2_ASSOC_CONF to the CONTROL FSM.
- Receive Event MT2_PEER_REASSOC_RSP (ASSOC-6). Call PeerAssocRspAction() to handle the re-association response frame from the BSSID and send a event MT2_ASSOC_CONF to the CONTROL FSM.
- Receive Event MT2_ASSOC_TIMEOUT (ASSOC-7). Call AssocTimeoutAction() to handle the (re)association response timeout action.

b. REASSOC_WAIT_RSP state:

- Receive Event MT2_MLME_ASSOC_REQ (REASSOC-1). Call InvalidStateWhenAssoc() to reset ASSOC FSM.
- Receive Event MT2_MLME_REASSOC_REQ (REASSOC-2). Call InvalidStateWhenReassoc() to reset ASSOC FSM.
- Receive Event MT2_MLME_DISASSOC_REQ (REASSOC-3). Call InvalidStateWhenDisassociate() to reset ASSOC FSM.
- Receive Event MT2_PEER_DISASSOC_REQ (REASSOC-4). Call PeerDisassocAction() to handle the disassociation response frame from the BSSID.

- Receive Event MT2_PEER_REASSOC_RSP (REASSOC-5). Call PeerReassocRspAction() to handle the reassociation response frame from the BSSID.
- Receive Event MT2_PEER_ASSOC_RSP (REASSOC-6). Call PeerReassocRspAction() to handle the association response frame from the BSSID.
- Receive Event MT2_REASSOC_TIMEOUT (REASSOC-7). Call ReassocTimeoutAction() to handle the reassociation response timeout action.

c. DISASSOC_WAIT_RSP state:

- Receive Event MT2_MLME_ASSOC_REQ (DISASSOC-1). Call InvalidStateWhenAssoc() to reset ASSOC FSM.
- Receive Event MT2_MLME_REASSOC_REQ (DISASSOC-2). Call InvalidStateWhenReassoc() to reset ASSOC FSM.
- Receive Event MT2_MLME_DISASSOC_REQ (DISASSOC-3). Call InvalidStateWhenDisassociate() to reset ASSOC FSM.
- Receive Event MT2_PEER_DISASSOC_REQ (DISASSOC-4). Call PeerDisassocAction() to handle the disassociation response frame from the BSSID.
- Receive Event MT2_DISASSOC_TIMEOUT (DISASSOC-5). Call DisassocTimeoutAction() to handle the disassociation response timeout action.

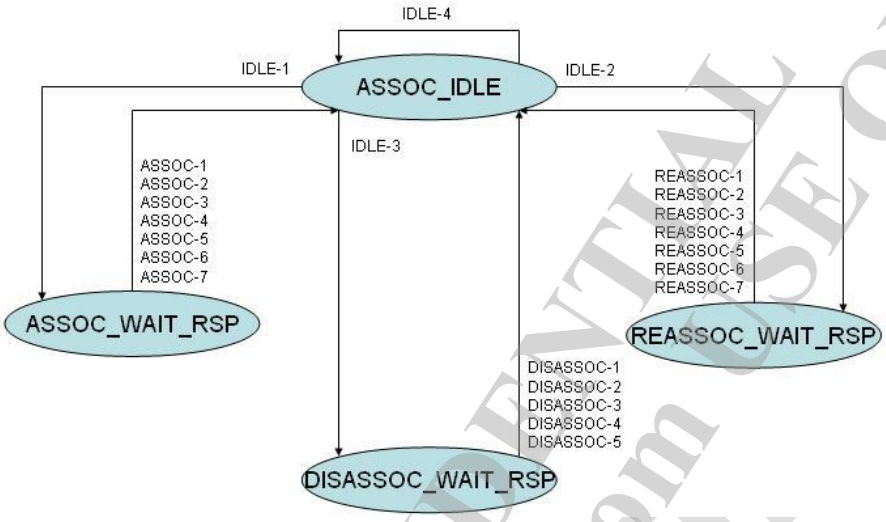


Figure 3-20 ASSOC FSM flow chart

3.7.3.4 CONTROL FSM

The Control FSM is responsible for AUTH/ASSOC FSM control.

STATE_MACHINE

Declare CntlMachine.

```
#define CNTL_IDLE                   0
#define CNTL_WAIT_DISASSOC          1
#define CNTL_WAIT_JOIN              2
#define CNTL_WAIT_REASSOC           3
#define CNTL_WAIT_START             4
#define CNTL_WAIT_AUTH              5
#define CNTL_WAIT_ASSOC             6
#define CNTL_WAIT_AUTH2             7
#define CNTL_WAIT_OID_LIST_SCAN     8
#define CNTL_WAIT_OID_DISASSOC      9

#define MT2_ASSOC_CONF              34
#define MT2_AUTH_CONF               35
#define MT2_DEAUTH_CONF             36
#define MT2_DISASSOC_CONF           37
#define MT2_REASSOC_CONF            38
#define MT2_PWR_MGMT_CONF           39
#define MT2_JOIN_CONF               40
#define MT2_SCAN_CONF               41
#define MT2_START_CONF              42
#define MT2_GET_CONF                43
#define MT2_SET_CONF                44
#define MT2_RESET_CONF              45
#define MT2_MLME_ROAMING_REQ        52
```

1. INIT & SET EVENT HANDLER (connect.c)

    a. CNTL_IDLE state - Call CntlIdleProc() to handle different requests.

- OID_802_11_SSID - Reconnect to a SSID. If we enable auto-connect function, we will call MlmeAutoReconnectLastSSID() in the STAMlmePeriodicExec() to issue the event request.
- OID_802_11_BSSID - Reconnect to a BSSID.
- OID_802_11_BSSID_LIST_SCAN - Do a scan.
- OID_802_11_DISASSOCIATE - Disassociate the AP
- MT2_MLME_ROAMING_REQ - Start roaming
- OID_802_11_MIC_FAILURE_REPORT_FRAME - security use

b. CNTL_WAIT_DISASSOC state - Call CntlWaitDisassocProc() to handle different request.

- MT2_DISASSOC_CONF - Only from DisassocTimeoutAction() and InvalidStateWhenDisassociate() of the ASSOC FSM. The handler will re-probe any available AP.

c. CNTL_WAIT_JOIN state - Call CntlWaitJoinProc() to handle different request.

- MT2_JOIN_CONF - Should be from the SYNC FSM. Find a available AP and start AUTH FSM.

d. CNTL_WAIT_REASSOC state - Call CntlWaitReassocProc() to handle different request.

- MT2_REASSOC_CONF - Should be from the ASSOC FSM. ASSOC is successfully and do link up.

e. CNTL_WAIT_START state - Call CntlWaitStartProc() to handle different request.

- MT2_START_CONF - IBSS built successfully.

f. CNTL_WAIT_AUTH state - Call CntlWaitAuthProc() to handle different request.

- MT2_AUTH_CONF - Should be from the AUTH FSM. AUTH is successfully and start ASSOC or REASSOC FSM.

g. CNTL_WAIT_AUTH2 state - Call CntlWaitAuthProc2() to handle different request.

- MT2_AUTH_CONF - Should be from the AUTH FSM. AUTH is successfully for WEP mode and start ASSOC or REASSOC FSM.

h. CNTL_WAIT_ASSOC state - Call CntlWaitAssocProc() to handle different request.

- MT2_ASSOC_CONF - Should be from the ASSOC FSM.ASSOC is successfully and do Link Up.

i. CNTL_WAIT_OID_LIST_SCAN state:

- MT2_SCAN_CONF - Resume TX/RX function after completed SCAN.

j. CNTL_WAIT_OID_DISASSOC state:

- MT2_DISASSOC_CONF - Only from DisassocTimeoutAction() and InvalidStateWhenDisassociate() of the ASSOC FSM. Disassociation successfully and do link down.

3.7.3.5 LINK UP FLOW

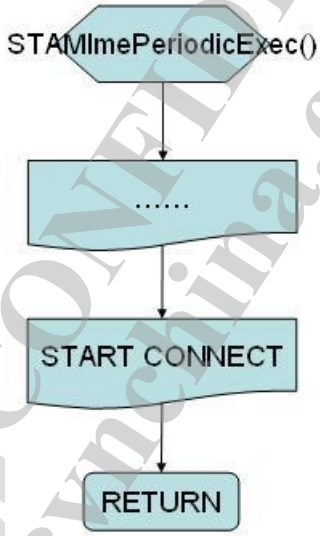


Figure 3-21 Link up flow chart A

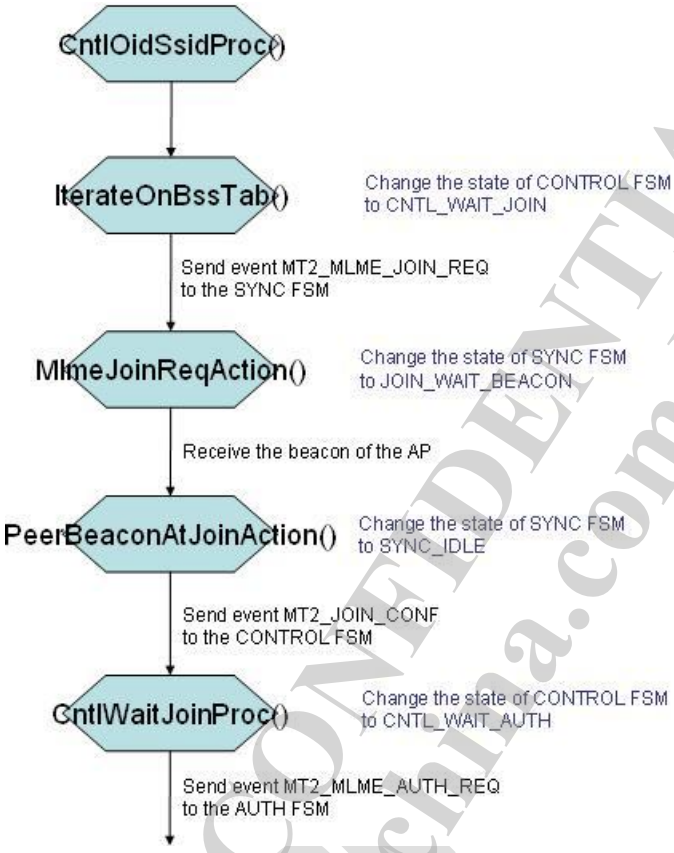
START CONNECT

Check if StaCfg.bAutoReconnect is TRUE;

If TRUE and no any AP information in the scan table, queue a event, MT2_MLME_SCAN_REQ, to the SYNC FSM and set the state of the CONTROL FSM to CNTL_WAIT_OID_LIST_SCAN.

If TRUE and at least one AP iformation in the scan table and the state of CONTROL FSM is CNTL_IDLE state, call MlmeAutoReconnectLastSSID() to send a event, OID_802_11_SSID, to CONTROL FSM to find if any expected AP is found.

Details



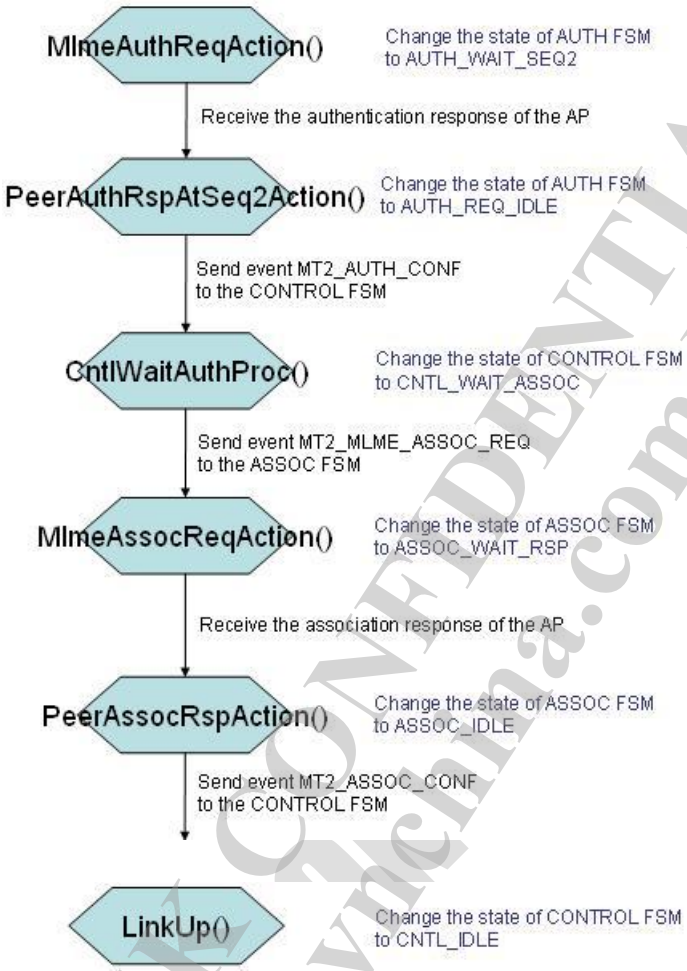


Figure 3-22 Link up flow chart B

4 MESH MODULE

4.1 Abbreviations and Acronyms

FSM	Finite State Machine
MAP	Mesh Access Point
MP	Mesh Point
MPP	Mesh Point collocated with a mesh Portal
TTL	Time to Live
UCG	Unified Channel Graph

4.2 Overview

4.2.1 Overview of the document

The document describes the basic design of Ralink mesh driver. We will review main mesh data structures and the interaction between different Data Structures in different stage. Refer to the IEEE 802.11s draft for detailed specifications.

4.2.2 Introduction of Mesh Network

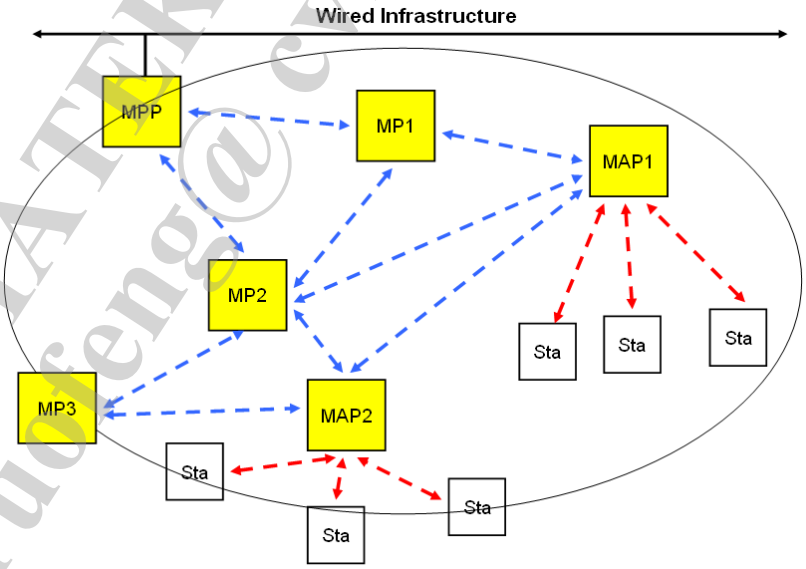


Figure 4-1 Example mesh network

An example mesh network is illustrated figure above. Mesh point (MPs) are QoS STAs that support mesh services, i.e., they participate in interoperable formation and operation of the mesh network. MP1 and MP2 has direct link to transfer packet. MP1 and MP3 has indirect link,MP2 and MAP2 can forward MP1's packet to MP3.An MP may be collocated with one or more other entities (e.g., APs, portals, etc.) The configuration of an MP that is connected with an Access Point is referred to as a Mesh Access Point (MAP). Such a configuration allows a single entity to logically provide both mesh functionalities and AP functionalities simultaneously. STAs associate with APs to gain access to the network. Only MPs participate in mesh functionalities such as path selection and forwarding. Mesh portals(MPPs) interface the network to other IEEE 802 LAN segments.

4.3 Mesh Data Structure

This section describes the data structures used by the mesh network.

4.3.1 _MESH_STRUCT

This structure is declared in _RTMP_ADAPTER and contains all data mesh needs.

1. Mesh configuration - e.g.,Mesh Seq, Mesh Path Selection protocol….
2. Mesh_IE - e.g., MeshId, Mesh Configuration IE,Mesh Security Capability IE….
3. Security
4. Connection,routing and Multicast flooding control table

   _MESH_NEIGHBOR_TAB
   _MESH_LINK
   _MESH_ENTRY_TABLE
   _MESH_ROUTING_TABLE
   _MESH_PROXY_ENTRY_TABLE
   _MESH_BMPKTSIG_TAB
   _MESH_MULTIPATH_ENTRY

Ralink Specific Features

1. UCHAR HostName[MAX_HOST_NAME_LEN] - 802.11s is layer 2 protocol, mesh entities can only identify each other by their MAC address, but the MAC address is an inconsequential

number. The user can configure the HostName to introduce itself to other mesh peers (e.g., “Tony’s Mesh” or “Mesh Server”)

2. BOOLEAN MeshOnly - In STA+Mesh mode, some STA UIs may ask the driver to scan frequently, even if the STA does not connect to the AP. This affects the mesh networks throughput. if the user sets MeshOnly to TRUE, the driver will ignore STA UI’s scan requests, and reply 0 bss.

3. BOOLEAN MeshAutoLink - If MeshAutoLink is TRUE, the mesh device will actively try to open a direct link with an MP. If MeshAutoLink is FALSE, the mesh device will passively listen to Peer Link Open from other MPs.

4.3.2 _MESH_NEIGHBOR_TAB

This structure is declared in _MESH_STRUCT and contains mesh Neighbor Information which listens to the beacons (e.g., MAC, MeshId, Channel, Channel Bandwidth, Routing algorithm.) The Neighbor Information will be updated when a mesh device receives the neighbor’s beacon. It has the most up to date neighbor information.

4.3.3 _MESH_LINK

This structure is declared in _MESH_STRUCT and contains MP ‘s data which mesh device is trying to connect to or mesh device already builds link with .

It contains the following MP data:

1. Mac, Channel, Bandwidth and Security which copy from _MESH_NEIGHBOR_TAB

2. Link_State, LocalLinkId and PeerLinkId which come from the process of building link.

4.3.4 _MESH_ENTRY_TABLE

This structure is declared in _MESH_STRUCT and contains destination of routing information. _MESH_ENTRY_TABLE will cooperate with _MESH_ROUTING_TABLE to finish routing information query and lookup.

There are 2 type destinations for routing information.

1. if PathReqTimerRunning is TRUE mesh device is sending Path Request to query routing information.

2. if PathReqTimerRunning is FALSE mesh device already has routing information of the destination, mesh device know which MP with direct link Can forward the packet to destination.

4.3.5 _MESH_ROUTING_TABLE

This structure is declared in _MESH_STRUCT and contains routing information that which MP with direct link can forward packet to mesh destination.

The destination in _MESH_ENTRY_TABLE might be:

1. MP with direct link

2. MP with indirect link (packet can be forwarded by other MP with direct link.)

3. The destination behind MP and the MP will forward packet to it.

4.3.6 _MESH_BMPKTSIG_TAB

This structure is declared in _MESH_STRUCT and maintains Mesh Sequence number of all MP .driver will drop duplicate broadcast and multicast packet and send Multi-Path notify to MP to stop the traffic from it. It will avoid broadcast and multicast flooding.

4.3.7 _MESH_MULTIPATH_ENTRY

This structure is declared in _MESH_STRUCT and maintains Multi-Path notify, if mesh device receive MP1 to stop broadcast and multicast traffic from MP2, it will not send packet from MP2 to MP1 for few secs.

4.3.8 _MAC_TABLE

This structure is declared in _RTMP_ADAPTER and contains mac, security and physical information. When a new entry is created in _MESH_LINK, it will create a mac entry in _MAC_TABLE.

4.4 Mesh Management Entity

This chapter describes the implementation of management entities used by the mesh management entity to perform management-layer dependent tasks provided in the Ralink mesh driver code.

4.4.1 Mesh State Machine

Mesh has 2 state machines

1. Mesh Control Management State Machine
2. Mesh Peer Link Management State Machine

4.4.1.1 Mesh Control Management State Machine

The Mesh Control Management State Machine controls the state of mesh device.

The State machine transfer is implemented in MeshCtrlStateMachineInit() in Mesh_ctrl.c

The Mesh Control Management State Machine uses the following four states:

1. IDLE – in the IDLE state, the finite state machine only responds to MESH_CTRL_JOIN which is generated by MeshUp.
2. DISCOVERY – in the DISCOVERY state, the finite state machine has actively issue a Scan Request to SYNC_STATE_MACHINE and transfer to ACTIVATED state after scan.
3. ACTIVATED – in ACTIVED state, the finite state machine will enable all mesh functions (e.g., generate beacon, establish link with mesh peer and run routing algorithm)
4. UCG – when the finite state machine is in ACTIVATED and receives UCG from MP, it will transfer to UCG state, send UCG to MP with direct link and switch to new channel at last.

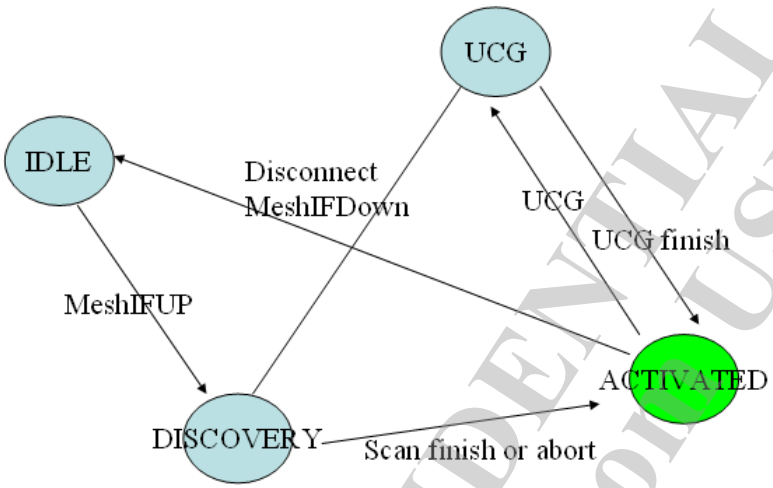


Figure 4-2 Mesh Control Management Protocol FSM

Due to special request, CMPC will skip DISCOVERY AND UCG state and related functions.

Peer Link Management State Machine

All MPs which a mesh device is trying to connect with, or have already opened a link, will have Peer Link Management State to control the link state. State machine transfer is implemented in MeshLinkMngStateMachineInit () in Mesh_link_mng.c.

Protocol

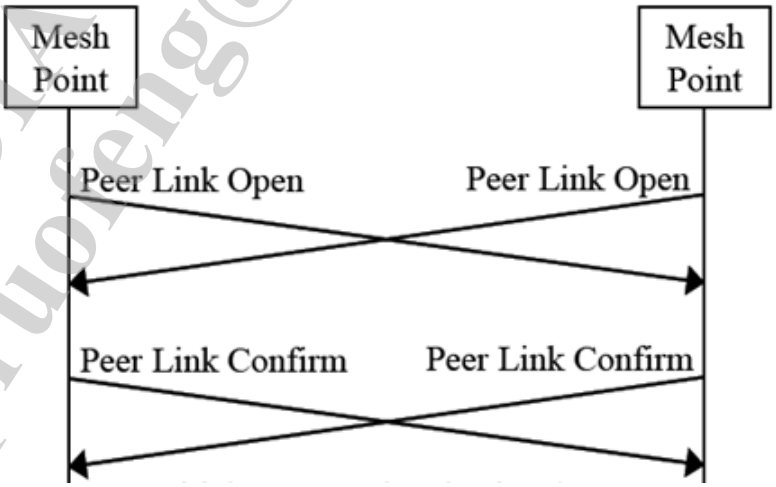


Figure 4-3 Peer Link Open and Confirm exchange

An MP must exchange 802.11s Peer Link Open and Peer Link Confirm to build direct link as figure above. Finite State Machine of Mesh Peer Link Management will handle the job to send Peer Link Open and Peer Link Confirm and process Mesh Peer's Peer Link Open and Peer Link Confirm. Refer to IEEE 802.11s draft 11B.3 for more detailed information.

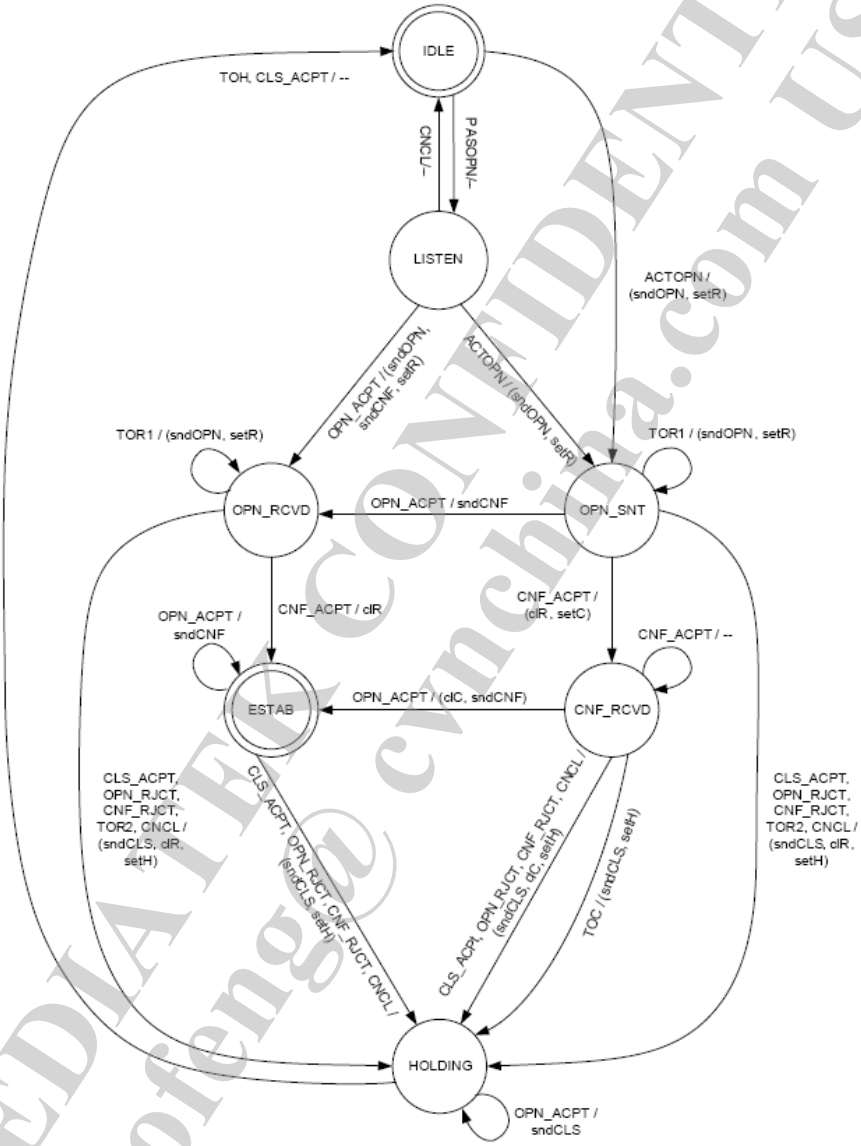


Figure 4-4 Mesh Peer Link Management FSM

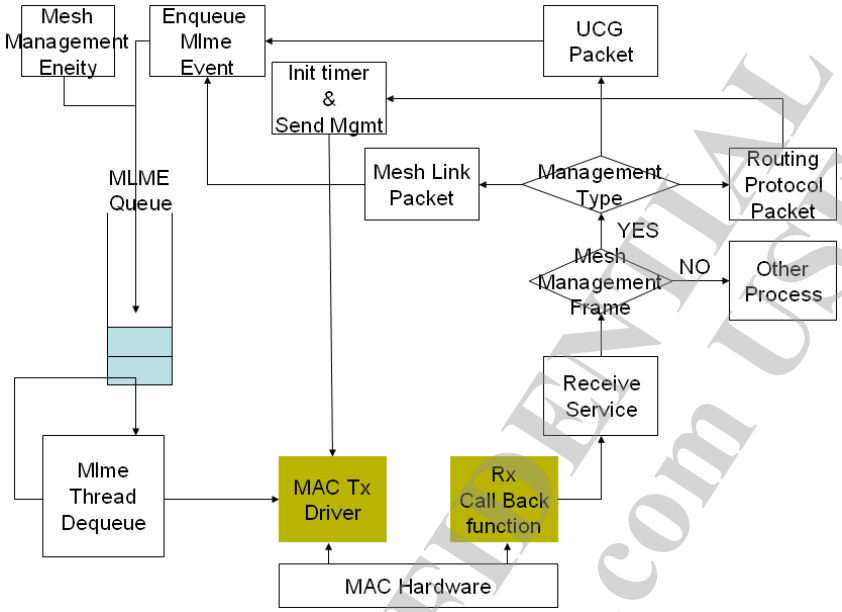


Figure 4-5 MLME flow chart

MLME Thread

Some activities in the MLME cannot be performed synchronously with the calling routine. Due to the design of USB, we cannot submit a URB in interrupt/tasklet. MLME Thread performs this task.

MESH MGMT Rx Path for STA+Mesh

1. Rx Call Back function is STARxDoneInterruptHandle()
2. Rx check is it Mesh Management Frame in STAHandleRxMgmtFrame()
3. Rx check which Mesh Management Type it is and call corresponding routine in MlmeHandleRxMeshFrame()
4. if the event is time limited, the routine sets a timer.
5. The routine might queue MLME Event to MLME Queue and wake up MLME Thread.
6. MLME Thread or timer might send packet.

4.5 Mesh Networking

This chapter describes Ralink's implementation of mesh. It includes the interaction between key data structures in each stage, and other Ralink specific features.

4.5.1 Mesh of Ralink Implementation and the interaction between the main data structures in each stage

This section describes the implementation of the mesh driver from;

1. Neighbor discover: Listen Stage
2. Building Link with MP: Link Stage
3. Querying routing path before send data packet: Routing Path Query Stage

4.5.1.1 Listen Stage

The mesh device listens to the beacons and update to _MESH_NEIGHBOR_TAB. It is implemented in PeerBeacon() in Sync.c.

4.5.1.2 Link Stage

Active Link

If MeshAutoLink in _MESH_STRUCT is TRUE, the mesh device will periodically select a candidate with the same MeshId and security from _MESH_NEIGHBOR_TAB to open a link.

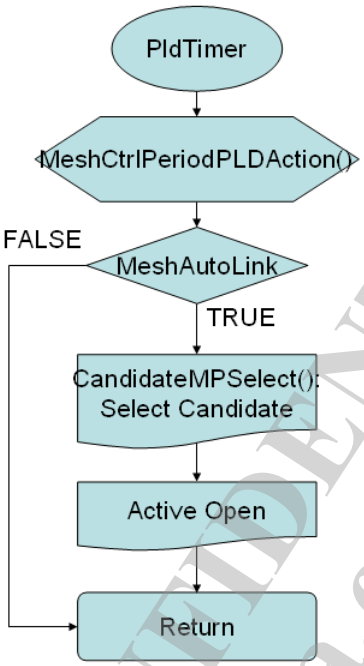


Figure 4-6 Link stage flow chart

Details

- PldTimer - It is 20secs periodic timer, it calls MeshCtrlPeriodPldAction()
- MeshCtrlPeriodPldAction - If MeshAutoLink is TRUE, it calls CandidateMPSelect(). If MeshAutoLInk is FALSE, it return…
- CandidateMPSelect - It search _MESH_NEIGHBOR_TAB for a corresponding MP which has the same MeshId, security, etc. It allocates an entry in _MESH_LINK and initiates the Mesh Peer Link Management State machine
- Active Open - Enqueue an Active Open Event to Mesh Peer Link Management State machine. MLME Thread dequeue the event, it send 802.11s Peer Link Open to MP and allocate a Entry in _MAC_TABLE.

Inactive Link

If MeshAutoLink in _MESH_STRUCT is FALSE, mesh Device passively wait MP's Peer Link Open Request.

4.5.1.3 Routing Path Query Stage

When a packet is sent to mesh Device from Os. Mesh Device will Query _MESH_ENTRY_TABLE and _MESH_ROUTING_TABLE. if there is no routing information in table, mesh Device will send Path Request to MP with direct link to query routing information.

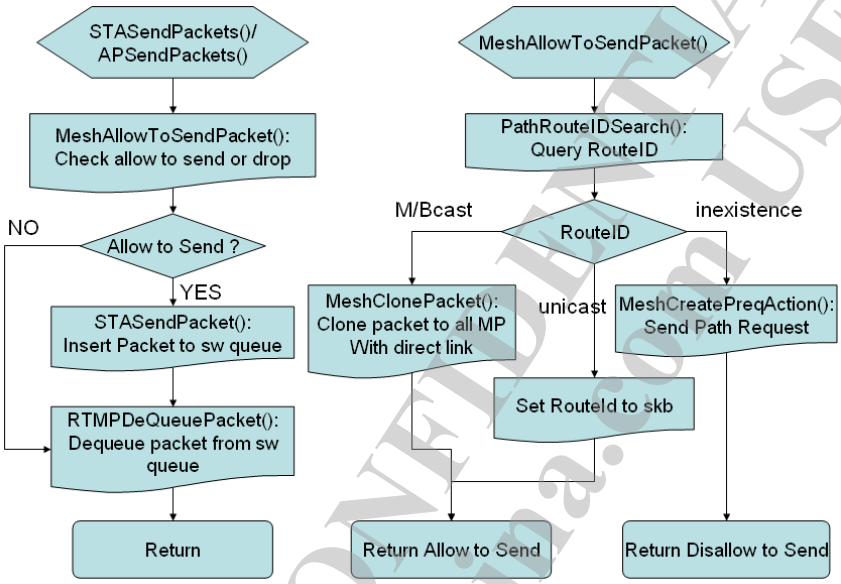


Figure 4-7 Routing path query stage flow chart

Details

1. STASendPackets/APSendPackets - Os call STASendPackets/APSendPackets to send Packet
2. MeshAllowToSendPacket - If packet come from Mesh Virtual Interface, MeshAllowToSendPacket will call PathRouteIDSearch to find RouteID
3. PathRouteIDSearch - Query RouteId:
    a. If RouteId is M/Bcast, it clones the packet to all MP with a direct link
    b. If there is RouteId, the mesh device know which MP with direct link can forward the packet and send the packet to the MP.
    c. If RouteId is inexistent, mesh Device free the skb and call MeshCreatePreqAction to query routing information
4. MeshCreatePreqAction - The function create 802.11s Path Request Frame and send it to all MP with direct link.

5. If mesh Device receive 802.11s Path Response Frame from other MP, it will update _MESH_ENTRY_TABLE and _MESH_ROUTING_TABLE.

4.5.1.3.1 _MESH_ENTRY_TABLE AND _MESH_ROUTING_TABLE RELATIONSHIP

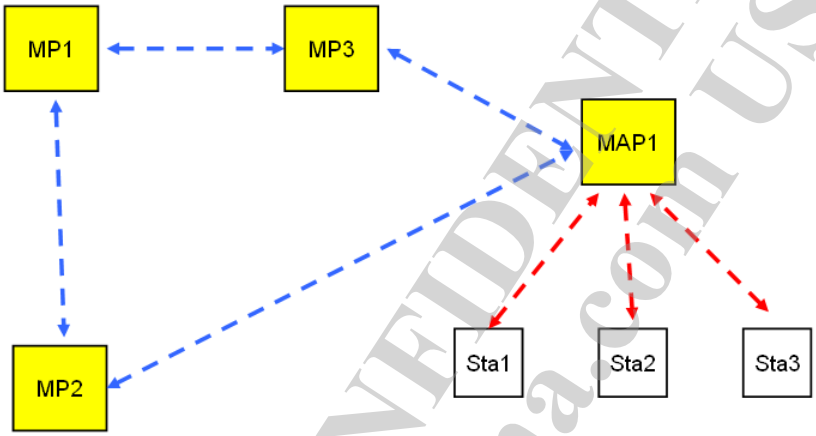


Figure 4-8 Mesh network topology

If the Mesh Network topology is similar to the one shown above, there are 3 MPs, 1 MAP and 3 STAs. The related list tables are shown below.

MeshLink[Idx]

Table 1 MP1's _MESH_LINK

Idx	Mesh Peer Link FSM	MP MAC Addr
0	ESTAB	MP2
1	ESTAB	MP3

Table 2 MP1's _MESH_ENTRY_TABLE

MAC Addr	Routing Path Idx
MP2	0
MP3	1
MAP1	2
Sta1	2
Sta2	2
Sta3	2

Table 3 MP1's _MESH_ROUTING_TABLE

Routing Path Idx	Mesh Destination	Next Hop	NextHopLinkID (Idx)
0	MP2	MP2	0
1	MP3	MP3	1
2	MAP1	MP3	1

Query Example:

If MP1 want to query Sta3 :

1. it will search _MESH_ENTRY_TABLE to find Routing Path Idx is 2.
2. It query _MESH_ROUTING_TABLE to find:
    a. MAP1 is the final destination in the mesh Network, MAP1 will forward the packet to Sta3.
    b. if a mesh Device wants to send a packet to MAP1, it must send the packet to MP3, which has a direct link with the mesh Device.

4.5.2 Multicast Flooding Control

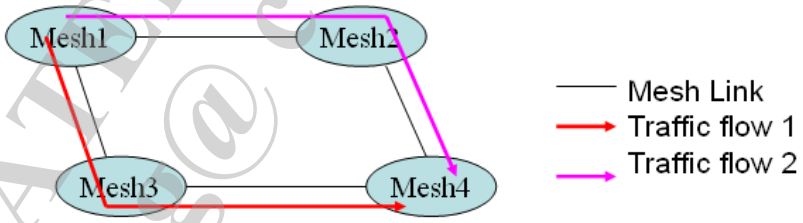


Figure 4-9 Multicast Flooding Control

mesh network can connect to other mesh networks. The same broadcast/multicast packet can be sent to the same receiver from different senders, as shown in the figure above. Traffic flow 1 and Traffic flow 2 degrade the throughput of mesh Network.

Multicast Flooding Control finds that Traffic flow 1 and Traffic flow 2 come from the same source, Mesh1. If Traffic flow 2 is more than Traffic flow 1, it will send a Multi path notification with Mesh1's MAC to Mesh2. Mesh2 will stop forwarding Mesh1's broadcast/multicast to Mesh4 for some time.

This feature is implemented by _MESH_BMPKTSIG_TAB and _MESH_MULTIPATH_ENTRY.

_MESH_BMPKTSIG_TAB records the Mesh Sequence Number of each Source Address. The check is performed in PktSigCheck(), which is called.

5 802.1X DAEMON

5.1 Overview

The chapter describes a user-space daemon which provides the IEEE 802.1X Authenticator feature. It relays the EAP message between the Supplicant and Authenticator Server(AS). For more detailed specifications, please refer to IEEE Std 802.1X-2004.

5.2 Introduction

IEEE 802.1X provides a port-based network access control. The Port Access Control offers a means of preventing unauthorized access by Supplicants to the services offered by that System, and also preventing a Supplicant from attempting to access an unauthorized System. Port Access Control also provides a means whereby a Supplicant system may prevent an unauthorized system from connecting to it. A Supplicant system can also make use of the outcome of the exchange to prevent unauthorized access.

Access control is achieved by the System enforcing authentication of Supplicants that attach to the System's controlled Ports; from the result of the authentication process, the System can determine whether or not the Supplicant is authorized to access its services on that controlled Port.

802.1X daemon is Authenticator role. An Authenticator PAE is responsible for enforcing the authentication of a Supplicant PAE that attaches to its controlled Port and for controlling the authorization state of the controlled Port accordingly.

5.3 File Structure

config.c/config.h : Query the configuration setting from wireless driver and parse them.

eapol_sm.c/eapol_sm.h : maintain the state machine for EAPoL frame.

eloop.c/eloop.h : Event loop for registering timeout calls, signal handlers, and socket read events.

ieee802_1x.c/ ieee802_1x.h : handle the incoming frame from Supplicants or AS.

md5.c/md5.h : provides MD5 hash and HMAC-MD5.

radius.c/ radius.h : RADIUS message generation and parsing functions.

radius_client.c/ radius_client/h : Communicate with AS (Radius server).

rt2860apd.c/rt2860apd.h : the entry point of daemon.

sta_info.c/ sta_info.h : the Supplicant information.

5.4 Work Flow

5.4.1 main routine

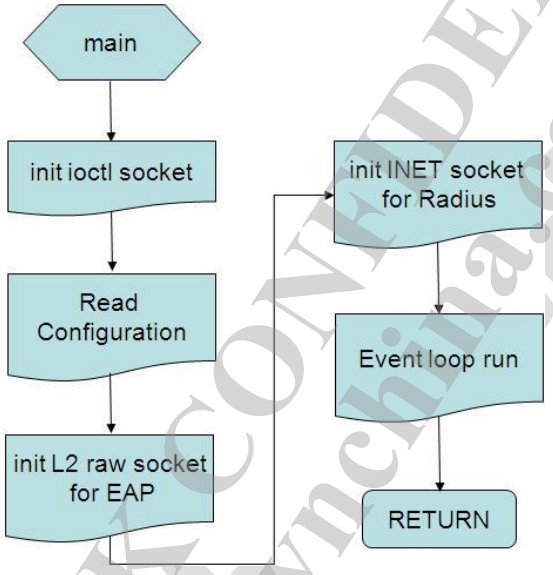


Figure 5-1 IEEE802.1x daemon INIT flow chart

Details

- init ioctl socket : initialize a socket for IOCTL. It's used to communicate with wireless driver
- read configuration : get the related configuration setting from wireless driver and parse them.
- init L2 raw socket for EAP : initialize sockets for receiving specific protocol packets and register them into event loop routine.
- init INET socket for Radius : initialize sockets for receiving Radius packets and register them into event loop routine.
- Event loop run : a task for registering timeout calls, signal handlers, and socket read events.

5.4.2 Event Loop

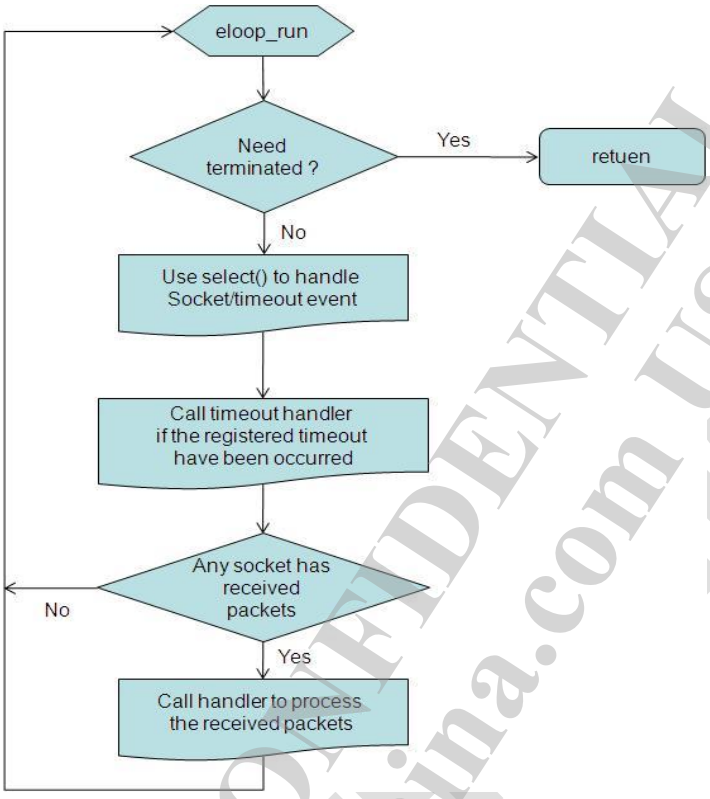


Figure 5-2 IEEE802.1x daemon EVENT flow chart

Details

- Use select() to handle the multiplex socket event and the registered timeout function.
- If the timer is expired, call the callback function of timeout.
- Use FD_ISSET to check the registered sockets to determine whether some sockets are ready to read. If the result is true, call the callback function to process them.

5.5 Ralink 802.1x daemon configuration setting

When the Ralink dot1x daemon starts, it reads the related configuration form wireless driver through IOCTL.

The configuration parameters are described as below,

5.5.1 Essential parameters

- own_ip_addr : specific the IP address of AP.
- RADIUS_Server : indicate the IP address of Radius server.
- RADIUS_Port : indicate the UDP port of Radius server

- RADIUS_Key : indicate the secret key of Radius server

5.5.2 Optional parameters

- session_timeout_interval : the expire of session disconnection. Its default value is 0.
- EAPifname : it is assigned as the binding interface for EAP negotiation. Its default value is “br0”.
- PreAuthifname : It is assigned as the binding interface for WPA2 Pre-authentication. Its default value is “br0”.

5.5.3 Dynamic WEP Supporting

In OPEN-WEP with 802.1x mode, the authentication process generates broadcast and unicast key. The unicast key is unique for every individual client so it is always generated randomly by 802.1x daemon. But the broadcast key is shared for all associated clients, it can be pre-set manually by users or generated randomly by 802.1x daemon.

Through the parameter "DefaultKeyID" and its corresponding parameter "KeyXStr"(i.e. X = the value of DefaultKeyID) in RT2860AP.dat, the 802.1x daemon would use it as the broadcast key material. But if the corresponding parameter "KeyXStr" is empty or unsuitable, the broadcast key would be generated randomly by the 802.1x daemon.

6 WPS - WI-FI PROTECTED SETUP

6.1 Definition

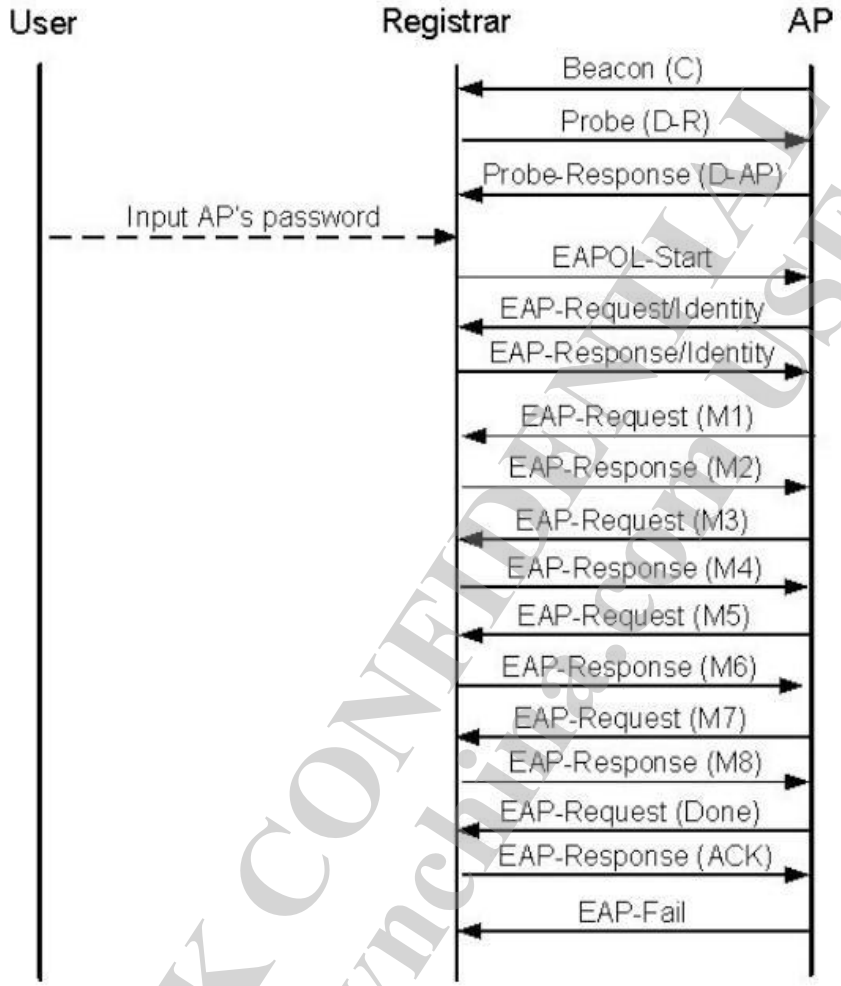
In-band	Data transfer by using EAP pakcets on WLAN communication channel
Out-of-band	Data transfer by using UPnP packets on a communication channel other than the WLAN.
Credential	A data structure issued by a Regisrar to an Enrollee.
Enrollee	A Deivce seeking to join a WLAN Domain.
Registrar	A entity with the authority to issue and revoke Domain Credentials. A Regisrar may be integrated into an AP, or it may be separate from the AP.
External Registrar	A Registrar for an AP's Domain that runs on a device separate from the AP
PBC	PushButton Configuration: A configuration method triggered by pressing a physical or logical button on the Enrollee and on the Registrar

6.2 Ralink WPS AP/STA Scope

6.2.1 EAP-based Setup of External Registrar

AP as Enrollee, get configuration from Registrar.

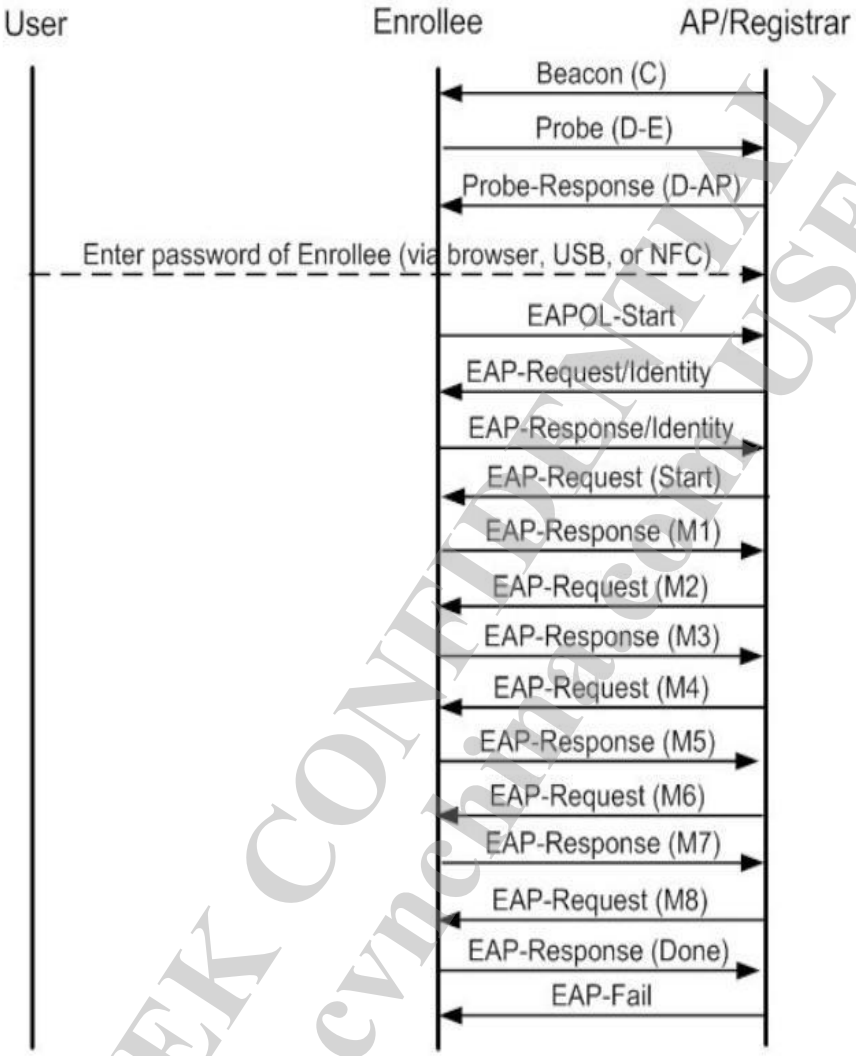
Station as Registrar provide configuration to AP (Enrollee).



6.2.2 In-bans Setup Using a Standalone AP/Registrar

AP with Registrar, provide the configuration to the Enrollee.

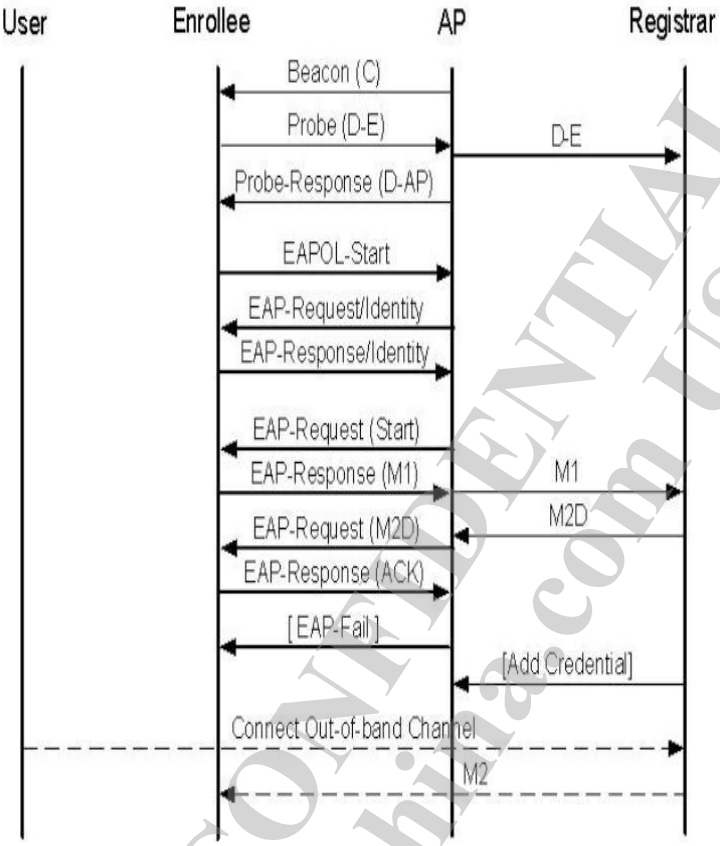
Station as Enrollee, get configuration from Registrar.



6.2.3 Out-of-band Setup Using External Registrar

AP as Proxy, transfer data for Enrollee and External Registrar.

Station as Enrollee, through AP to get configuration from External Registrar.



6.3 WPS Packet Format

Vendor-Id: 0x00372A (WFA SMI code)

Vendor-Type: 0x0000 0001 (SimpleConfig)

Op-Code:

0x01 : WSC_Start

0x02 : WSC_ACK

0x03 : WSC_NACK

0x04 : WSC_MSG

0x05 : WSC_Done

0x06 : WSC_FRAG_ACK

6.4 WPS Packets Routing Decision in AP Driver

6.4.1 Flowchart

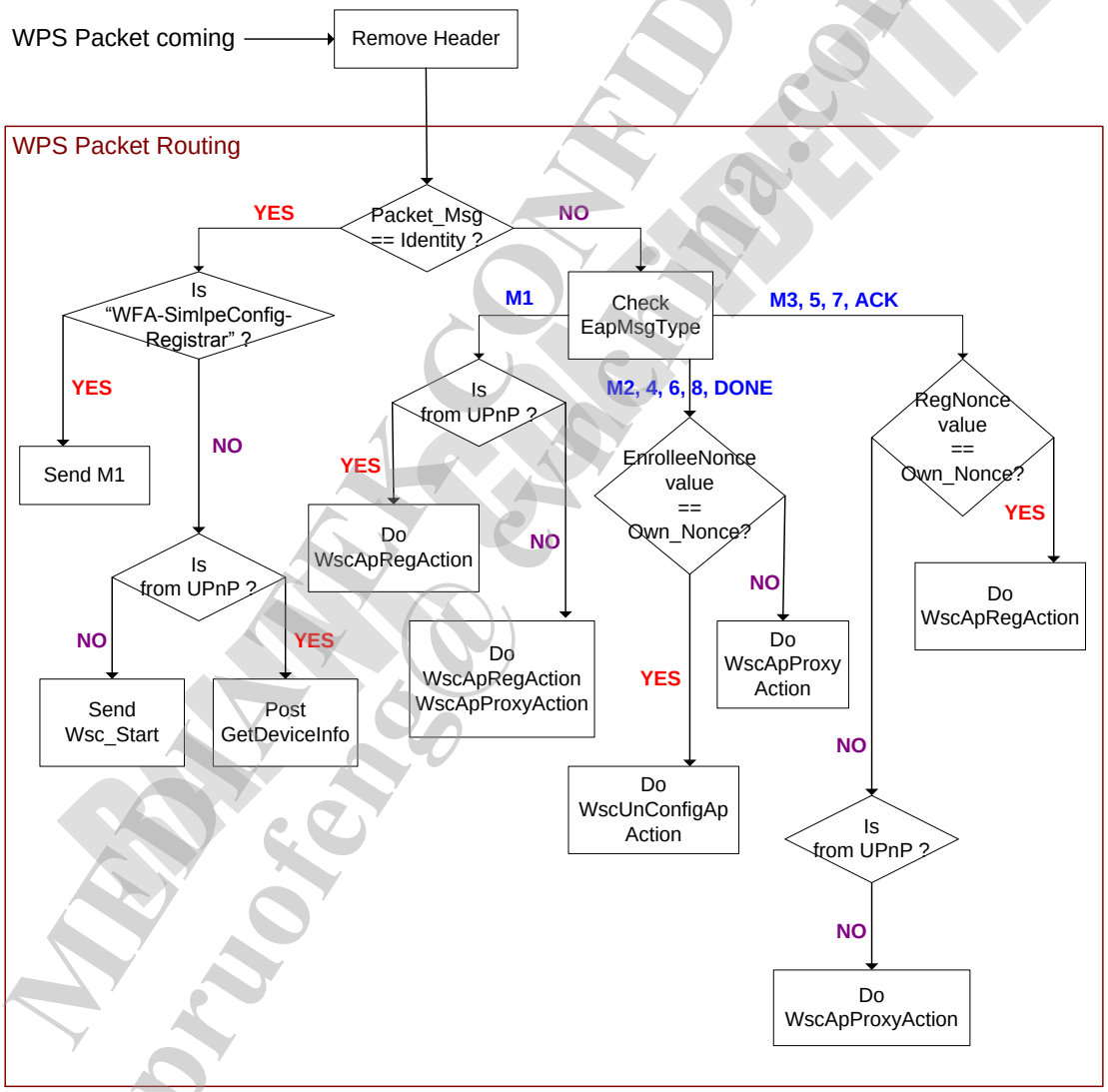


Figure 6-1 WPS Packet Routing flow chart

6.4.2 Pseudo Code

```
if ( PacketMsg == "WFA-SimpleConfig-Registrar" )
            MsgType = WSC_MSG_EAP_REG_RSP_ID;
else if ( PacketMsg == "WFA-SimpleConfig-Enrollee" )
            MsgType = WSC_MSG_EAP_ENR_ENR_ID;
else
{
            find WSC_IE_MSG_TYPE in PacketMsg;
            MsgType = MsgValue;
}

if ( MsgType == WSC_MSG_EAP_REG_RSP_ID )
     Do Send WscMessageM1;
else if ( MsgType == WSC_MSG_ENR_ENR_ID )
{
      if ( !bUPnPMsg )
            Do send WscEapMsgStart;
      else
            Do post GetDeviceInfo;
}
else if ( MsgType == WSC_MSG_M1 )
{
      Do WscApRegistrarAction;
      if ( !bUPnPMsg )
              Do WscApProxyAction;
}
else if ( MsgType == WSC_MSG_M3 ||
      MsgTYpe == WSC_MSG_M5 ||
      MsgType == WSC_MSG_M7 ||
      MsgType == WSC_MSG_WSC_DONE )
{
      if ( CheckRegNonce() )
            Do WscApRegistrarAction;
      else
      {
            if ( !bUPnPMsg )
                  Do WscApProxyAction;
      }
```

```
}
else if ( MsgType == WSC_MSG_M2 ||
          MsgTYpe == WSC_MSG_M4 ||
          MsgType == WSC_MSG_M6 ||
          MsgType == WSC_MSG_M8 ||
MsgType == WSC_MSG_WSC_ACK )
{
          if ( CheckEnrNonce() )
               Do WscUnConfiguredApAction;
          else
               Do WscApProxyAction;
}
```

6.5 WPS process in STA Driver

When WPS AP is in WPA/WPA2 or WPAPSK/WPA2PSK, RSN/SSN cannot present in association frame.

STA has to decide to be Enrollee or Registrar before triggered to do WPS process with WPS AP.

6.5.1 In-band Enrollee

Step 1)

Scan and checks Beacon from WPS AP. If user chooses PBC method, STA cannot do WPS process with WPS AP when more than one Beacon of WPS AP with PBC method.

Step 2)

Sends Probe Request to WPS AP.

Step 3)

After association success, sends EAPOL-Start to WPS AP.

Step 4)

After receiving EAP-Request/ID from WPS AP, sends EAP-Response/ID(WFA-SimpleConfig-Enrollee-1-0) to WPS AP.

Step 5)

After receiving EAP-Request (WSC_START) from WPS AP, starts to do EAP messages exchange with WPS AP. (STA sends EAP-Response packets)

Step 6)

After EAP messages exchange done, re-connects to WPS AP with new configuration and reset WSC_STATE to OFF.

6.5.2 In-band Registrar

Step 1)

Scans and checks Beacon from WPS AP. If user chooses PBC method, STA cannot do WPS process with WPS AP when more than one WPS AP broadcast Beacon with PBC method.

Step 2)

Sends Probe Request to WPS AP

Step 3)

After association success, sends EAPOL-Start to WPS AP.

Step 4)

After receiving EAP-Request/ID from WPS AP, sends EAP-Response/ID(WFA-SimpleConfig-Registrar-1-0) to WPS AP.

Step 5)

Starts to do EAP Messages exchange with WPS AP. (STA sends EAP-Response packets)

Step 6)

After EAP messages exchange done, re-connect to WPS AP and resets WSC_STATE to OFF.

6.6 WPS File Description

wsc.h – WPS Data Structure Definitions

wsc_tlv.h – WPS Data Element Definitions

wsc.c – WPS Function State Machine

wsc_tlv.c – WPS Messages Build/Process

6.6.1 Data Structure

WSC_CTRL : WPS Control Block, maintain WPS related information, main items are as below,

WscConfMode : 0x0 – WPS Disabled

0x1 - Enrollee

0x2 – Proxy

0x4 – Registrar

(0x1 | 0x2) – Enrollee + Proxy (Only AP supports this mode)

(0x1 | 0x4) – Enrollee + Registrar (Only AP supports this mode)

(0x2 | 0x4) – Proxy + Registrar (Only AP supports this mode)

(0x1 | 0x2 | 0x4) – Enrollee + Proxy + Registrar (Only AP supports this mode)

WscMode : 1 - PIN

2 – PBC

WscConfStatus : 1 un-configured (When STA is enrollee, STA is always un-configured)

2 configured

WscConfigMethods : Registrar support list. The List is bitwise.

PBC : 0x0080

Lable : 0x0004

Display : 0x0008

WscStatus : For user to monitor the status

WscState : WPS Protocl State

WscPinCode : Record the UI's PIN code input when we are registrar

WscEnrolleePinCode : Recored Device own PIN code

WSC_LV_INFO : Structure to store Simple Config Attributes Info.

WSC_PROFILE : WPS configured profile (has 8 credentials).

WSC_CREDENTIAL : WPS configured credential

6.6.2 WPS FSM

6.6.2.1 WPS roles at AP

1. Enrollee

2. Internal Registrar

3. Proxy

6.6.2.2 WPS roles at Station

1. Enrollee

2. EAP-based External Registrar

6.6.2.3 Running Scenarios

1. Adding an AP as an Enrollee to a station as an External Registrar (EAP)

[External Registrar]<----EAP--->[Enrollee_AP]

2. Adding a station as an Enrollee to an AP with built-in Registrar (EAP)

[Registrar_AP]<----EAP--->[Enrollee_STA]

3. Adding an Enrollee with External Registrar (UPnP/EAP)

[External Registrar]<----UPnP--->[Proxy_AP]<---EAP--->[Enrollee_STA]

6.6.2.4 WPS major APIs

WscEAPAction – Implement WPS Packets Routing rules.

WscEapEnrolleeAction – Process WPS packets from Registrar.

WscEapRegistrarAction – Process WPS packets from Enrollee.

WscEapApProxyAction – Process WPS packets between Enrollee and External Registrar.

Wsc2MinsTimeOutAction – 2 minutes timeout function after starting WPS process.

WscEAPOLTimeOutAction – Timeout checking for each WPS packet.

write_dat_file_thread – Kernel thread to write DAT file after WPS process completes

WscInit – Init value of WPS control block

WscStop – Stop WPS process

BuildMessageMx – Build WPS EAP message packet (M1, M2, …M8)

ProcessMessageMx – Process WPS EAP message packet (M1, M2, …, M8) from peer.

6.6.3 WPS IOCTLs

6.6.3.1 IOCTLs for AP

Set_AP_WscConfMode_Proc – Set WscConfMode of AP(Enrollee, Registrar, Proxy)

Set_AP_WscConfStatus_Proc – Set configure status of AP (configured or un-configured)

Set_AP_WscMode_Proc – Set mode of AP (using PIN or PBC)

Set_AP_WscPinCode_Proc – Set Pin Code of Enrollee to AP when AP is Registrar

Set_AP_WscGetConf_Proc – Trigger AP to do WPS process

RTMPIoctlWscProfile – Get WPS profile

6.6.3.2 IOCTLs for STA

Set_WscConfMode_Proc – Set WscConfMode of Station (Enrollee or Registrar)

Set_WscSsid_Proc – Set SSID of WPS AP when using PIN method.

Set_WscMode_Proc – Set mode to do WPS with WPS AP (PIN or PBC)

Set_WscPinCode_Proc – Set Pin Code of Enrollee to Station when Station is Registrar

Set_WscGetConf_Proc – Trigger STA to do WPS process

7 APPENDIX A: SUPPORTED PARAMETERS IN RT2870STA.DAT

MeshId

Description: Set the Mesh point ID

Value Type: Alphanumeric Characters

Valid Range: <32 characters

MeshAutoLink

Description: Enable or disable automatically adding the link between MPs.

Value Type: Unsigned Character

Valid Range: 0-1

0: Disable auto link action

1: Enable auto link action

MeshAuthMode

Description: Set the Mesh authentication mode.

Value Type: Text

Valid Range: OPEN, WPANONE

MeshEncrypType

Description: Set the Mesh encryption type.

Value Type: Text

Valid Range: NONE, WEP, TKIP, AES

MeshWPAKey

Description: Set the Mesh key in TKIP/AES mode.

Value Type: ASCII or hexadecimal characters

Valid Range: 8-63 ASCII characters or 64 hexadecimal characters

Default Value:

MeshDefaultKey

Description: Set the Mesh WEP default key index.

Value Type: Unsigned Character

Valid Range: 1-4 characters

Default Value:

MeshWEPKey

Description: Set the Mesh key in WEP mode.

Value Type: ASCII or hexadecimal characters

Valid Range: 5 or 13 ASCII characters or 10 or 26 hexadecimal characters

8 APPENDIX B: IWPRIV MESH0 SET COMMAND FOR MESH

iwpriv mesh0 set [parameters]=[Value]

Execute one iwpriv/set command simultaneously

MeshId

Description: Set the Mesh point ID

Value Type: Alphanumeric Characters

Valid Range: <32 characters

MeshHostName

Description: Set the MP's name for identification.

Value Type: ASCII characters

Valid Range: <32 ASCII characters

MeshAutoLink

Description: Enable or disable automatic adding of a link between MPs.

Value Type: Unsigned Character

Valid Range: 0-1

0: Disable, 1: Enable

MeshAddLink

Description: Manually add a neighbor MP to the link table.

Value Type: MAC address

MeshDelLink

Description: Manually delete a neighbor MP from the link table.

Value Type: MAC address

MeshAuthMode

Description: Set the Mesh authentication mode.

Value Type: Text

Valid Range: OPEN, WPANONE

MeshEncrypType

Description: Set the Mesh encryption type.

Value Type: Text

Valid Range: NONE, WEP, TKIP, AES

MeshDefaultKey

Description: Set the Mesh WEP default key index.

Value Type: Unsigned Character

Valid Range: 1-4 characters

MeshWEPKey

Description: Set the Mesh key in WEP mode.

Value Type: ASCII or hexadecimal characters

Valid Range: 5 or 13 ASCII characters or 10 or 26 hexadecimal characters

MeshWPAKey

Description: Set the Mesh key in TKIP/AES mode.

Value Type: ASCII or hexadecimal characters

Valid Range: 8-63 ASCII characters or 64 hexadecimal characters

8.1 Example

Example 1

Config Mesh Point set auto link disable

➔ iwpriv mesh0 set MeshAutoLink=0

Example 2

Config Mesh Point Add Mesh Point neighbor MP.

➔ iwpriv mesh0 set MeshAddLink=00:10:60:21:88:61

Example 3

Config Mesh Point Delete Mesh Point neighbor MP.

➔ iwpriv mesh0 set MeshDelLink=00:10:60:21:88:61

Example 4

Set Mesh Point Encryption Type as OPEN-NONE

➔ iwpriv mesh0 set MeshAuthMode=OPEN

➔ iwpriv mesh0 set MeshEncrypType=NONE

➔ iwpriv mesh0 set MeshId=CMPC-mesh

Example 5

Set Mesh Point Encryption Type as OPEN-WEP

➔ iwpriv mesh0 set MeshAuthMode=OPEN

➔ iwpriv mesh0 set MeshEncrypType=WEP

➔ iwpriv mesh0 set MeshDefaultkey=1

➔ iwpriv mesh0 set MeshWEPKEY=1234567890

➔ iwpriv mesh0 set MeshId=CMPC-mesh

Example 6

Set Mesh Point Encryption Type as WPANONE-TKIP

➔ iwpriv mesh0 set MeshAuthMode=WPANONE

➔ iwpriv mesh0 set MeshEncrypType=TKIP

➔ iwpriv mesh0 set MeshId=CMPC-mesh

➔ iwpriv mesh0 set MeshWPAKEY=12345678

➔ iwpriv mesh0 set MeshId=CMPC-mesh

Example 7

Set Mesh Point Encryption Type as WPANONE-AES

➔ iwpriv mesh0 set MeshAuthMode=WPANONE

➔ iwpriv mesh0 set MeshEncrypType=AES

➔ iwpriv mesh0 set MeshId=CMPC-mesh

➔ iwpriv mesh0 set MeshWPAKEY=12345678

➔ iwpriv mesh0 set MeshId=CMPC-mesh

9 APPENDIX C: PARAMETERS FOR IOCTL

ALL IOCTLs and Data Structures are defined in oid.h

(1) Set Raw Data with Flags For Mesh Link

IOCTL Function	
Set Raw Data by I/O Control Interface with Flags	
Function Type	IOCTL

RT_OID_802_11_MESH_SECURITY_INFO
 	
Set Mesh Security Information.
- Encryption type
- Key material
- Default key index
	printf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(MESH_SECURITY_INFO));
	wrq.u.data.length = sizeof(MESH_SECURITY_INFO);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_MESH_SET_SECURITY;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_MESH_ID
 	
Set Mesh-ID.
	printf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, MAX_MESH_ID_LEN);
	wrq.u.data.length = MAX_MESH_ID_LEN;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_MESH_MESH_ID;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_MESH_AUTO_LINK
 	
Enable automatic candidate MP selection.
0: disable.
1: enable.
	Sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(unsigned char));
	wrq.u.data.length = sizeof(unsigned char);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_MESH_AUTO_LINK;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_MESH_ADD_LINK
 	
Request to add a Mesh link manual.
	Sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, MAC_ADDR_LEN);
	wrq.u.data.length = MAC_ADDR_LEN;
	wrq.u.data.pointer = data;

	wrq.u.data.flags = RT_OID_802_11_MESH_ADD_LINK; ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_MESH_DEL_LINK
 	
Rquest to delete a Mesh link manual.
	Sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, MAC_ADDR_LEN);
	wrq.u.data.length = MAC_ADDR_LEN;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_MESH_DEL_LINK;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_MESH_CHANNEL
 	
Set Mesh Channel .
	Sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(unsigned char));
	wrq.u.data.length = sizeof(unsigned char);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_MESH_CHANNEL;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_MESH_HOSTNAME
 	
Set MeshHostName
	printf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, MAX_HOST_NAME_LEN);
	wrq.u.data.length = MAX_HOST_NAME_LEN;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_MESH_HOSTNAME;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_MESH_ONLY_MODE
 	
Enable mesh only function.
0: disable.
1: enable.
	Sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(unsigned char));
	wrq.u.data.length = sizeof(unsigned char);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_MESH_ONLY_MODE;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_MESH_FORWAR
 	
Enable mesh forwarding
0: disable.
	Sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(unsigned char));
	wrq.u.data.length = sizeof(unsigned char);

1: enable.	wrq.u.data.pointer = data;
	wrq.u.data.flags = RT_OID_802_11_MESH_FORWARD;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_MESH_CHANNEL_BW
 	
set mesh channel bandwidth
0: 20
1: 40
	Sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(unsigned char));
	wrq.u.data.length = sizeof(unsigned char);
	wrq.u.data.pointer = data;
	wrq.u.data.flags=RT_OID_802_11_MESH_CHANNEL_BW;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

RT_OID_802_11_MESH_CHANNEL_OFFSET
 	
set mesh channel OFFSET
0: below
1: above
	Sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(unsigned char));
	wrq.u.data.length = sizeof(unsigned char);
	wrq.u.data.pointer = data;
	wrq.u.data.flags=RT_OID_802_11_MESH_CHANNEL_OFFSET;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

(2) Get Raw Data with Flags For Mesh Link

IOCTL Function	
Get Raw Data by I/O Control Interface with Flags	
Function Type	IOCTL

OID_802_11_MESH_SECURITY_INFO
 	
Get Mesh Security Information.
- Encryption type
- Key material
- Default key index
	printf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(MESH_SECURITY_INFO));
	wrq.u.data.length = sizeof(MESH_SECURITY_INFO);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_MESH_SECURITY_INFO;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_MESH_ID
 	
Get Mesh-ID.
	printf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, MAX_MESH_ID_LEN);
	wrq.u.data.length = MAX_MESH_ID_LEN;

	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_MESH_ID;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_MESH_AUTO_LINK
 	
Get current Auto Link setting.
0: disable.
1: enable.
	sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(unsigned char));
	wrq.u.data.length = sizeof(unsigned char);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_MESH_AUTO_LINK;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_MESH_LINK_STATUS
 	
List all Mesh Links status.
0: idle.
1: connected.
	sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(MESH_LINK_INFO));
	wrq.u.data.length = sizeof(MESH_LINK_INFO);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_MESH_LINK_STATUS;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_MESH_LIST
 	
List all Neighbor MPs.
	sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(MESH_NEIGHBOR_INFO));
	wrq.u.data.length = sizeof(MESH_NEIGHBOR_INFO);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_MESH_LIST;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_MESH_ROUTE_LIST
 	
List route path on the Mesh Point
	sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(RT_MESH_ROUTE_TABLE));
	wrq.u.data.length = sizeof(RT_MESH_ROUTE_TABLE);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_MESH_ROUTE_LIST;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_MESH_CHANNEL
 	
Get Mesh Channel
	sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(unsigned char));

	wrq.u.data.length = sizeof(unsigned char);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_MESH_CHANNEL;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_MESH_ONLY_MODE
 	
Get current Mesh-Only setting.
0: disable.
1: enable.
	sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(unsigned char));
	wrq.u.data.length = sizeof(unsigned char);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_MESH_ONLY_MODE;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_MESH_DEVICENAME
 	
Get Mesh Device Name
	printf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, IFNAMSIZ);
	wrq.u.data.length = IFNAMSIZ;
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_MESH_DEVICENAME;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_MESH_CHANNEL_BW
 	
Get Mesh Channel band width
0: 20
1: 40
	sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(unsigned char));
	wrq.u.data.length = sizeof(unsigned char);
	wrq.u.data.pointer = data;
	wrq.u.data.flags = OID_802_11_MESH_CHANNEL_BW;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);

OID_802_11_MESH_CHANNEL_OFFSET
 	
Get Mesh Channel Offset
0: below
1: above
	sprintf(name, "mesh0");
	strcpy(wrq.ifr_name, name);
	memset(data, 0, sizeof(unsigned char));
	wrq.u.data.length = sizeof(unsigned char);
	wrq.u.data.pointer = data;
	wrq.u.data.flags=OID_802_11_MESH_CHANNEL_OFFSET;
	ioctl(socket_id, RT_PRIV_IOCTL, &wrq);