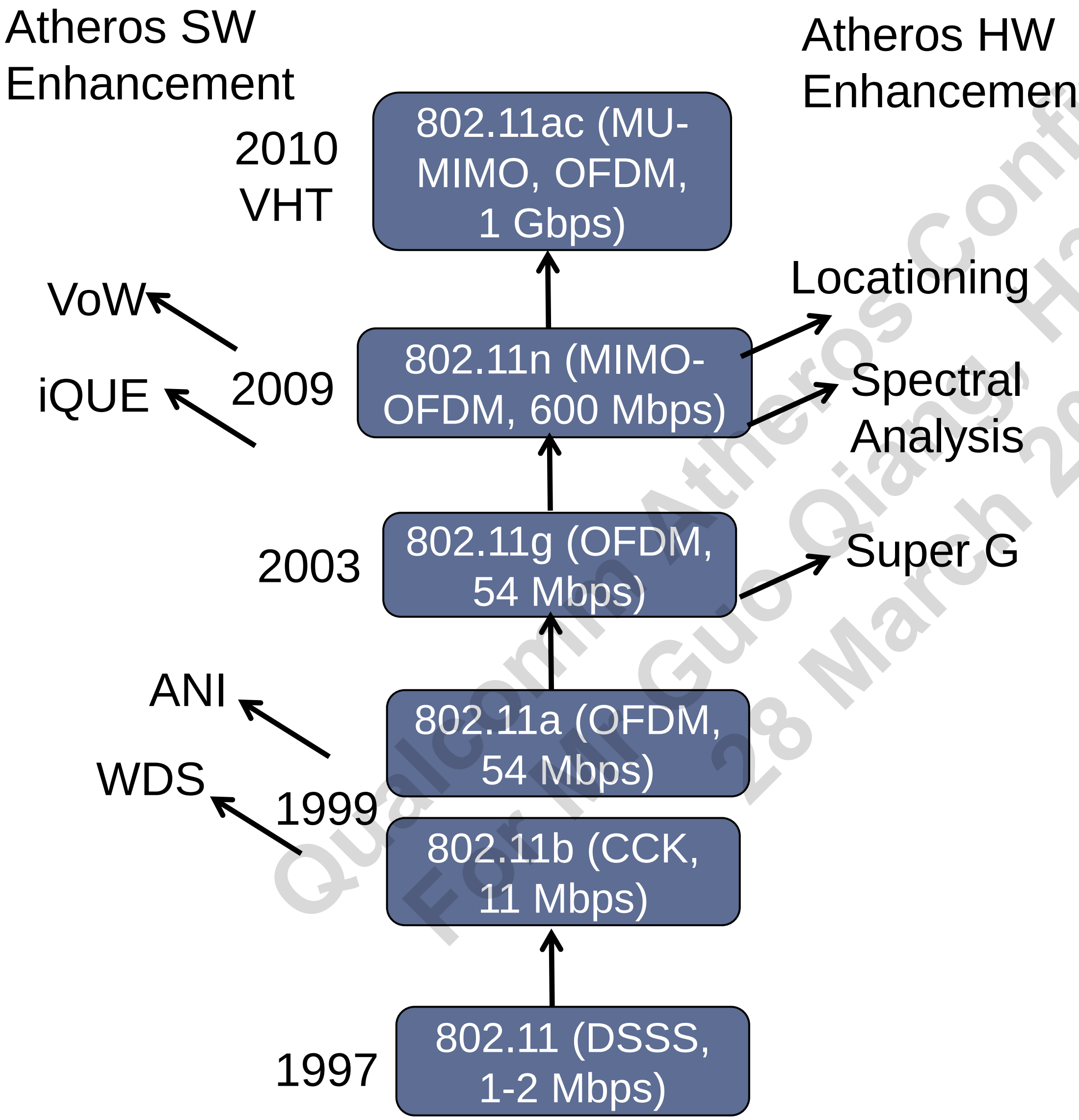
11ac Driver Usage

Content

1. Introduction
   - 1.1 802.11ac
   - 1.2 System Overview
   - 1.3 Release Documents
2. How to Build
   - 2.1 Firmware Build
   - 2.2 Host Driver Build
3. AP/STA Op
   - 3.1 Key Parameters
   - 3.2 Configuration Options
   - 3.3 AP Operation
   - 3.4 STA Operation
   - 3.5 iperf Parameters
4. Driver Debugging
   - 4.1 Target Data Path Statistics
   - 4.2 Host Data Path Statistics
   - 4.3 Target Mem/Reg Dump/change
   - 4.4 FW Debug Log
   - 4.5 11ac Tools

1.1 802.11ac



Atheros SW Enhancement

Atheros HW Enhancement

2010 VHT

802.11ac (MU-MIMO, OFDM, 1 Gbps)

VoW

iQUE

2009

802.11n (MIMO-OFDM, 600 Mbps)

Locationing

Spectral Analysis

2003

802.11g (OFDM, 54 Mbps)

Super G

ANI

WDS

1999

802.11a (OFDM, 54 Mbps)

802.11b (CCK, 11 Mbps)

1997

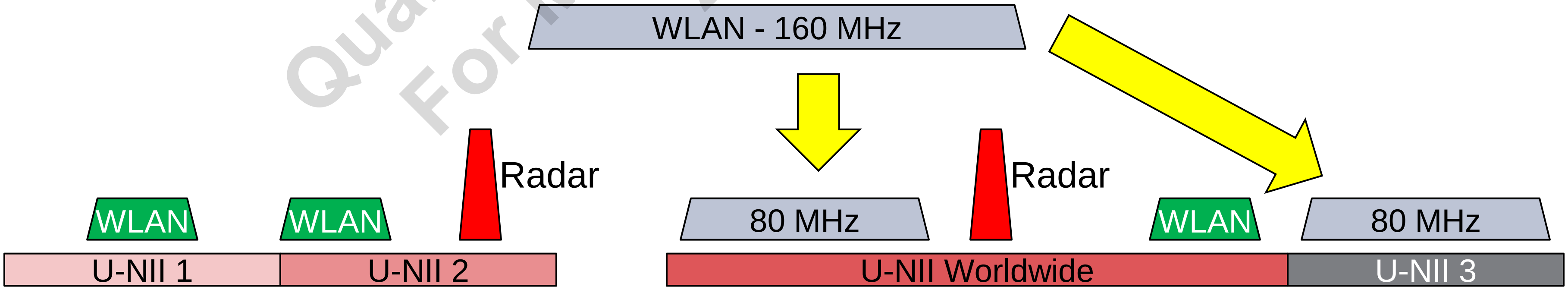
802.11 (DSSS, 1-2 Mbps)

11ac is the next major PHY & MAC upgrade after 11n

- Higher throughput while maintaining compatibility with 11a/n (5 GHz)
- Wider bandwidth
- Higher MIMO order
- Downlink multi-user MIMO
- 256-QAM
- Unified transmit beamforming scheme
- MAC enhancements
- Power saving features

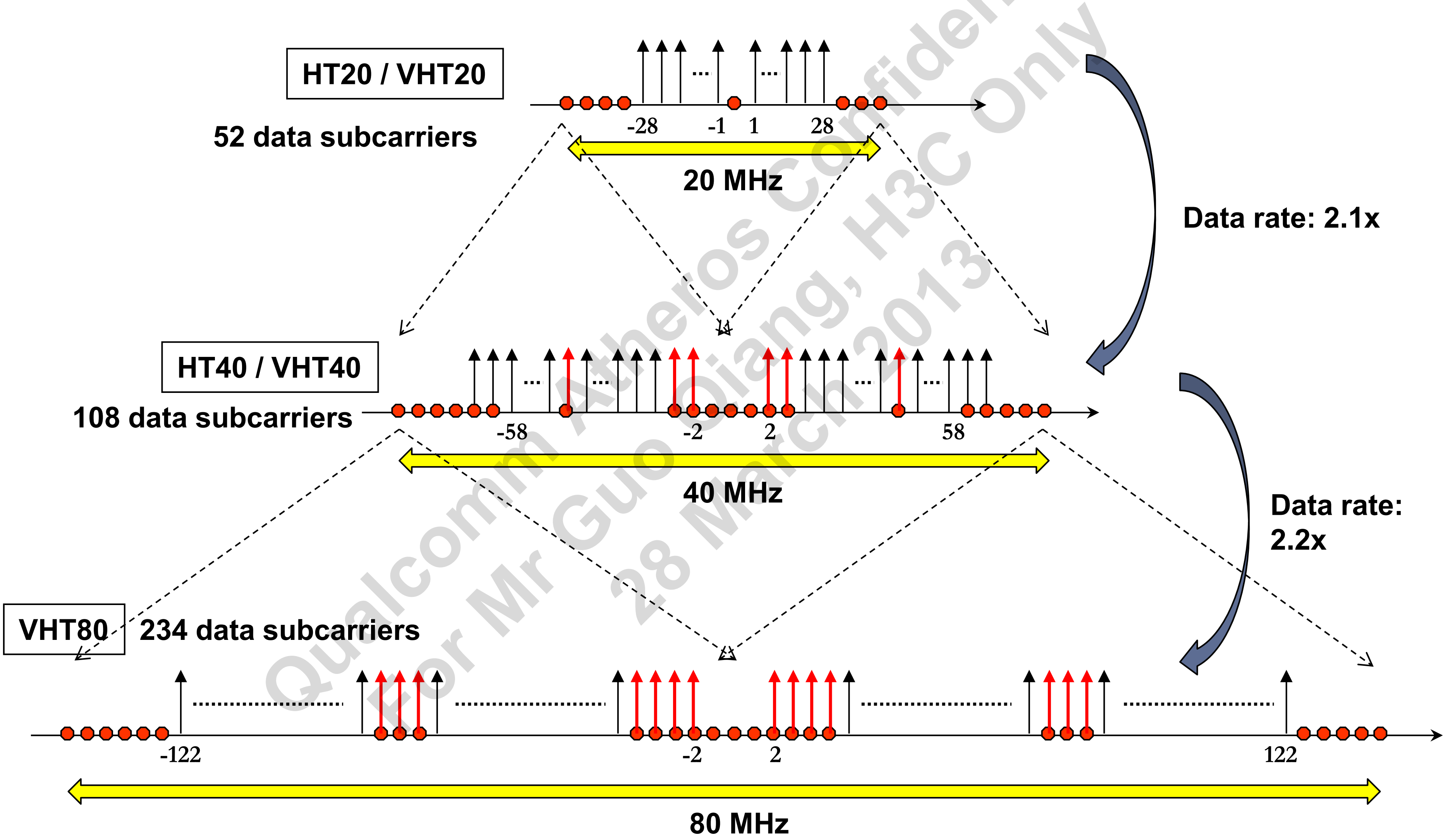
1.1.1 Wider Bandwidth

- Increase signal bandwidth
  - Effectively increase data rate with least complexity
  - For example, following modes can achieve > 1 Gbps TCP/IP throughput (assuming 70% protocol overhead): 160 MHz, 3-stream, 64-QAM 2/3; 80 MHz, 4-stream, 256-QAM 3/4
- More difficult to find one contiguous chunk of spectrum to use as BW increases
  - Spectrum divided by regulatory body into slices that do not easily accommodate wide bandwidth transmissions
  - Existing narrower (20/40 MHz) WLAN and radio devices (e.g. radar)
- Noncontiguous transmission
  - Split up the signal into two frequency segments: Higher probability of finding multiple narrower interference free channels than finding one wide interference free channel
  - Limit to two non-contiguous segments for reasonable tradeoff between complexity and flexibility
  - The two frequency segments are used synchronously: Both in TX or both in RX mode
  - Signal on the two segments are destined to the same receiver(s)



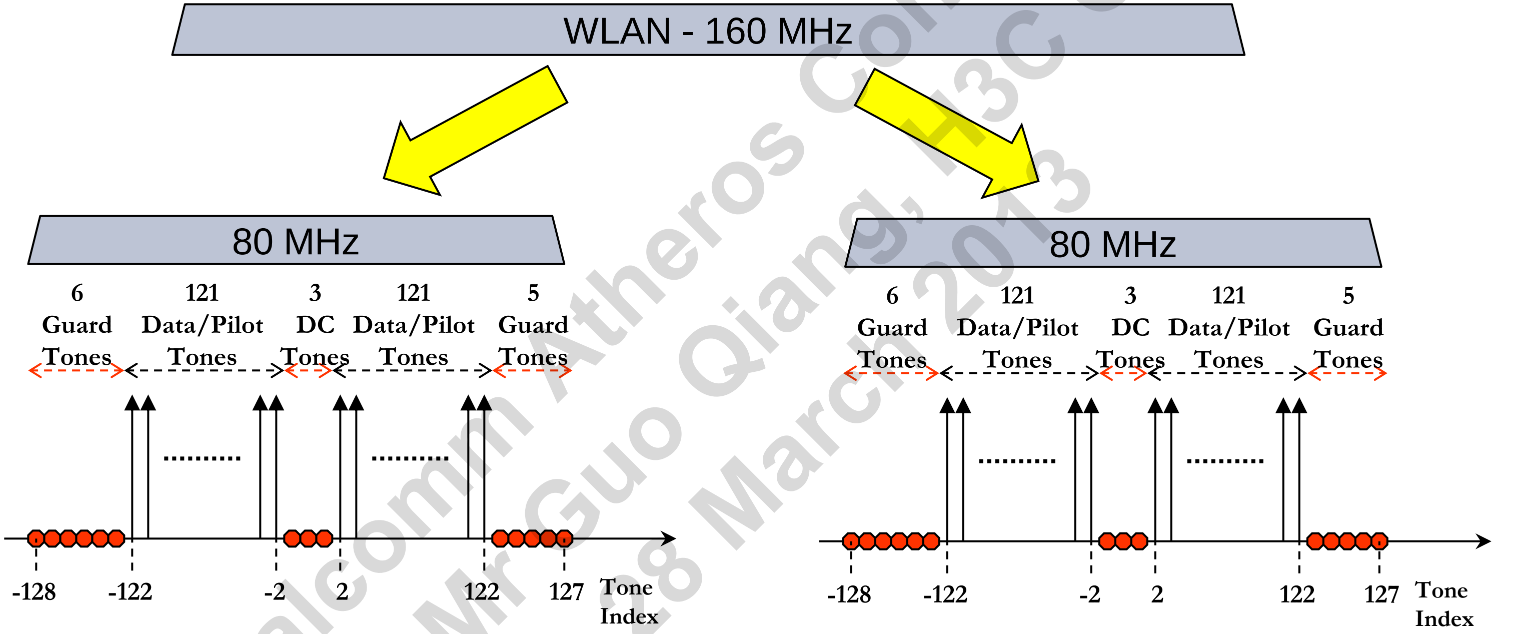
80 MHz Tone Allocation

- Additional subcarriers utilized to achieve > 2x increase in data rate



160 MHz Tone Allocation: Noncontiguous

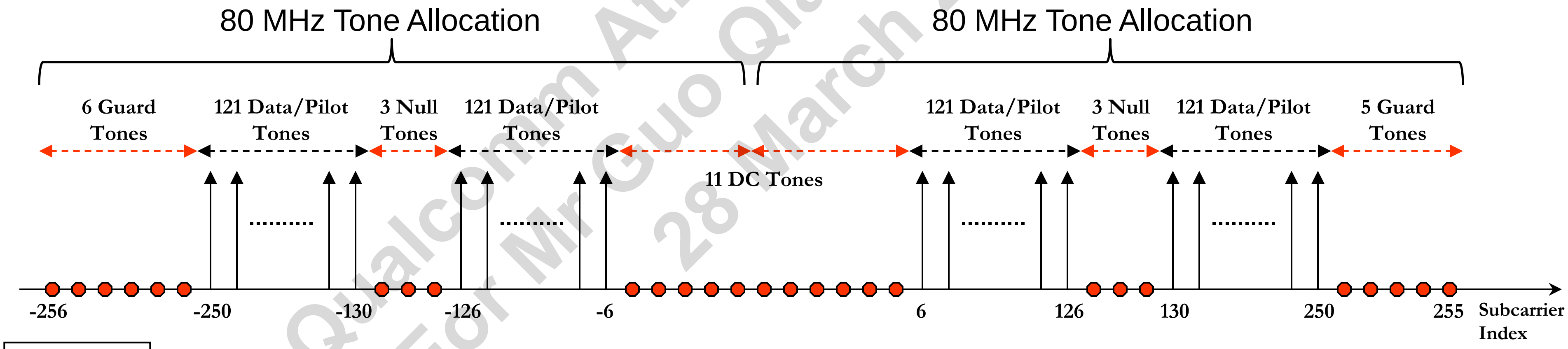
- Each frequency segment shall follow the 80 MHz tone allocation



VHT80+80	468 data subcarriers

160 MHz Tone Allocation: Contiguous

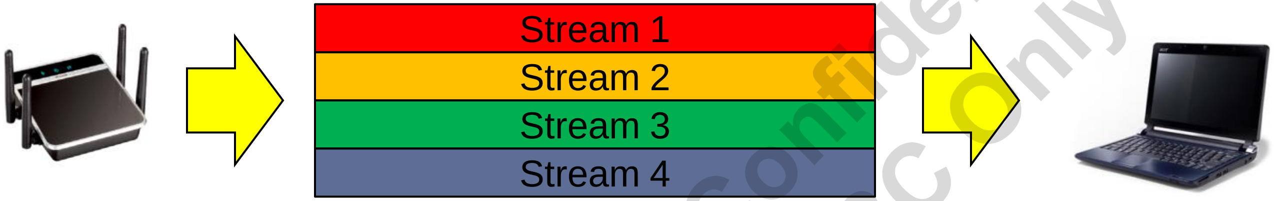
- Tone allocation equivalent to placing two 80 MHz tone allocations next to each other in frequency
  - Allows contiguous and noncontiguous 160 MHz devices to interoperate when the two segments of the noncontiguous devices are placed next to each other in frequency
  - Single PHY rate table for both contiguous and noncontiguous 160 MHz



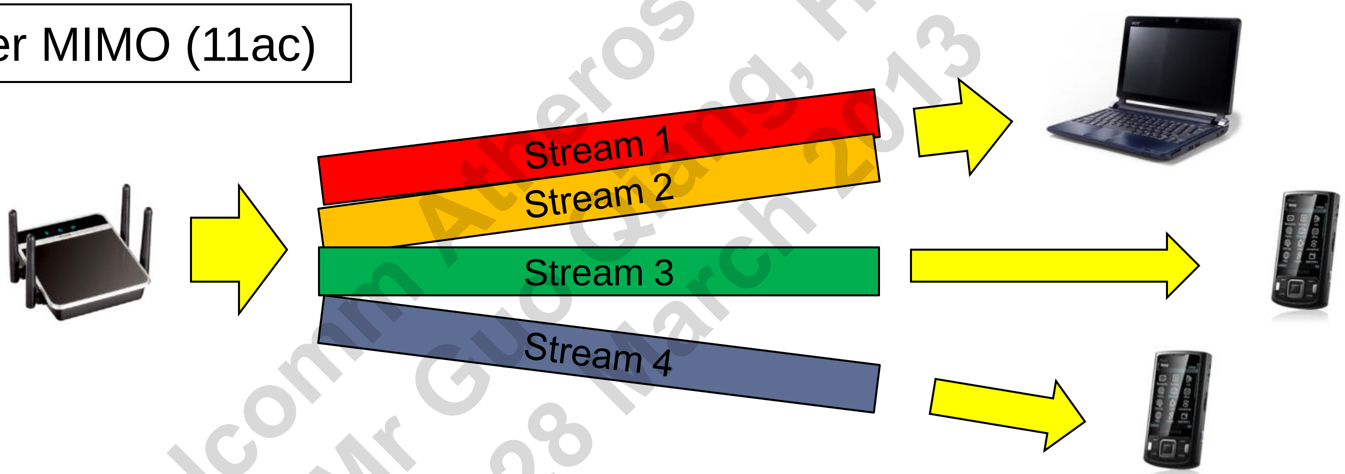
VHT160 **468 data subcarriers**

1.1.2 Downlink Multi-User MIMO (DL MU-MIMO)

Single-User MIMO (11n)



Multi-User MIMO (11ac)



- AP transmits to multiple client devices *simultaneously*
- Beamforming and nulling based on electronically steered antenna array are used to isolate data streams between the clients
- Allows sophisticated AP to maintain high total downlink throughput even when surrounded by simple (inexpensive) clients

1.1.3 256-QAM

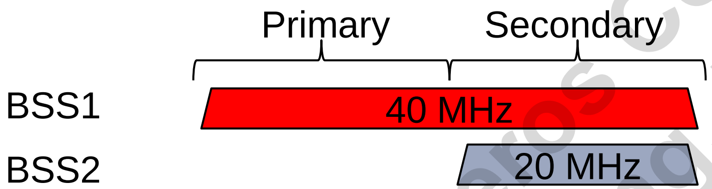
MCS	Nss=1			Nss=3		
	40 MHz	80 MHz	160 MHz	40 MHz	80 MHz	160 MHz
BPSK 1/2	15.0	32.5	65.0	45.0	97.5	195.0
QPSK 1/2	30.0	65.0	130.0	90.0	195.0	390.0
QPSK 3/4	45.0	97.5	195.0	135.0	292.5	585.0
16-QAM 1/2	60.0	130.0	260.0	180.0	390.0	780.0
16-QAM 3/4	90.0	195.0	390.0	270.0	585.0	1170.0
64-QAM 2/3	120.0	260.0	520.0	360.0	780.0	1560.0
64-QAM 3/4	135.0	292.5	585.0	405.0	-	1755.0
64-QAM 5/6	**150.0**	325.0	650.0	**450.0**	975.0	1950.0
256-QAM 3/4 ***New!***	180.0	390.0	780.0	540.0	1170.0	2340.0
256-QAM 5/6	200.0	433.3	866.7	600.0	1300.0	-

- Data rate assuming short GI
- Nss: Number of spatial streams
- Some BW+MCS combinations are not allowed to simplify TX flow
  - Nss=3 with 64-QAM 3/4, Nss=3 with 256-QAM 5/6

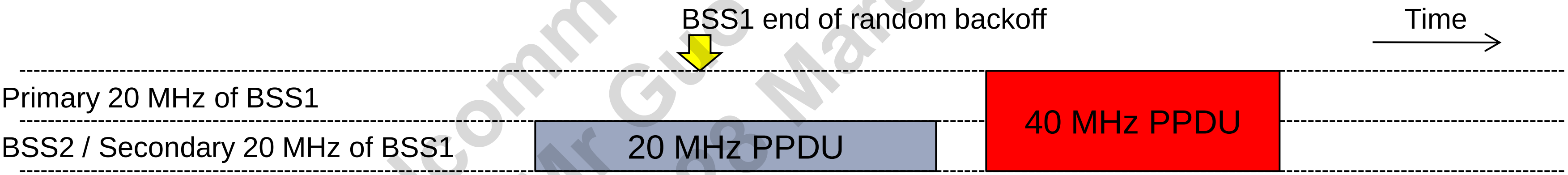
1.1.4 Other Features

Dynamic Bandwidth Transmission

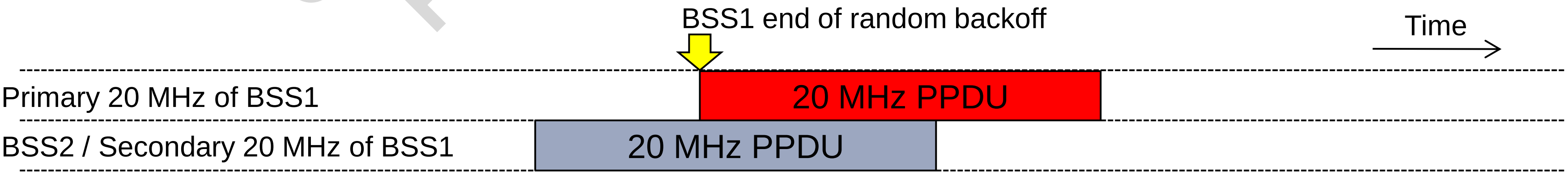
- 11n
  - 40 MHz BSS overlapping with 20 MHz BSS in secondary channel
  - Secondary channel is busy when random backoff in the primary channel is finished



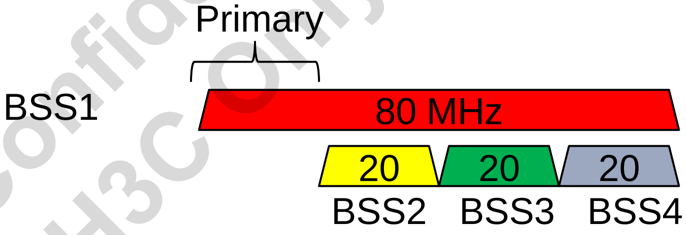
– Static BW transmission: Wait until entire 40 MHz is available



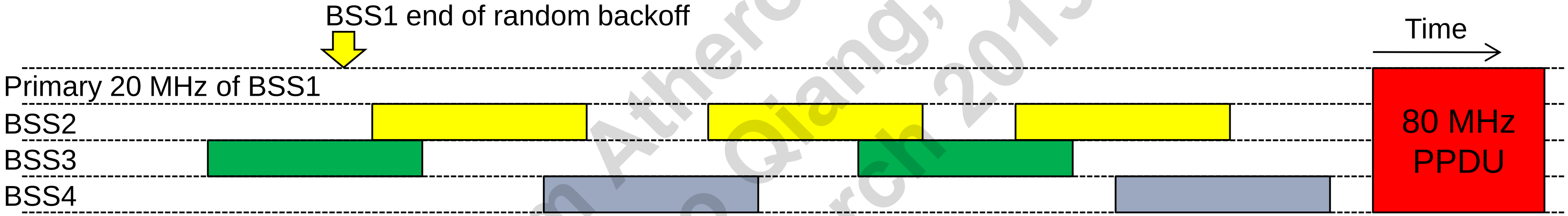
– Dynamic BW transmission: If secondary 20 MHz is busy, transmit 20 MHz PPDU



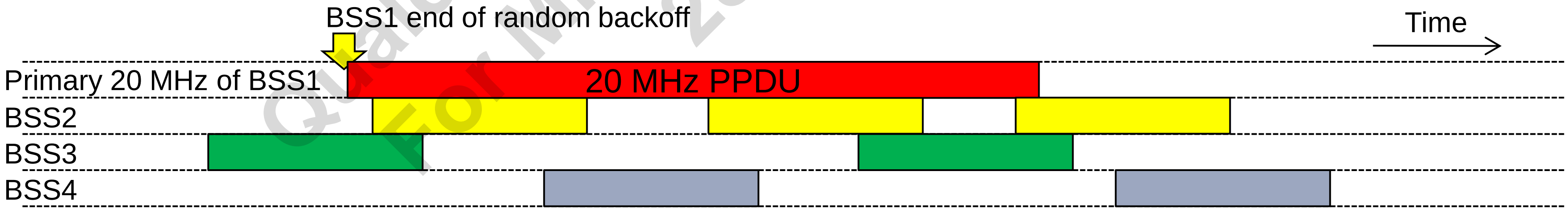
- Situation is worse for 80 and 160 MHz
  - Could possibly overlap with multiple narrower BSSs
  - May need to wait very long for the entire 80/160 MHz to be free
  - → Significant throughput degradation



Static BW Transmission



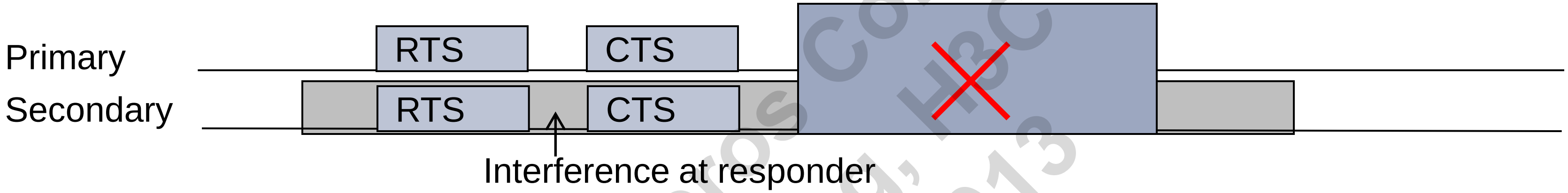
Dynamic BW Transmission



- Support of dynamic BW transmission is important in 11ac

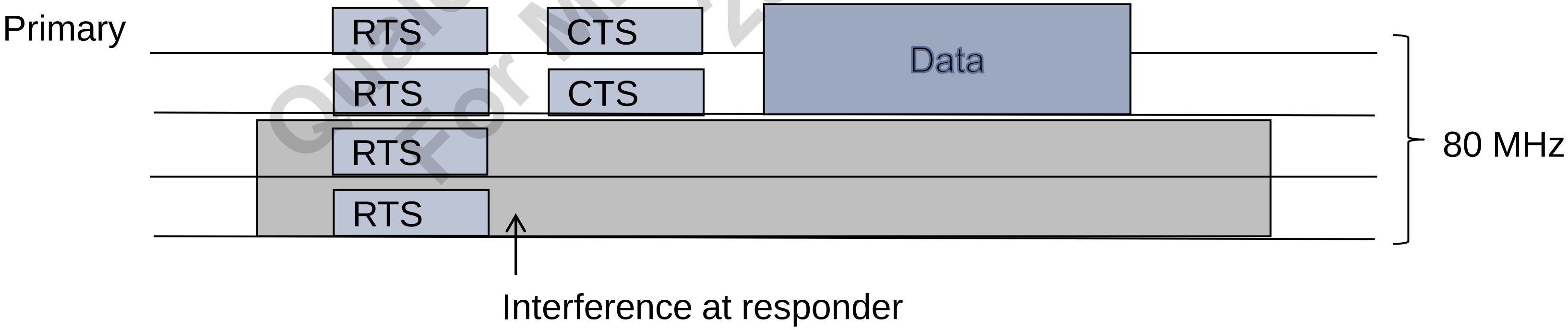
RTS/CTS

- 11n
  - Responder sends CTS on all subchannels
  - If an initiator receives CTS on the primary channel, it responds with Data on all subchannels
  - Does not account for hidden node collisions on the secondary channel



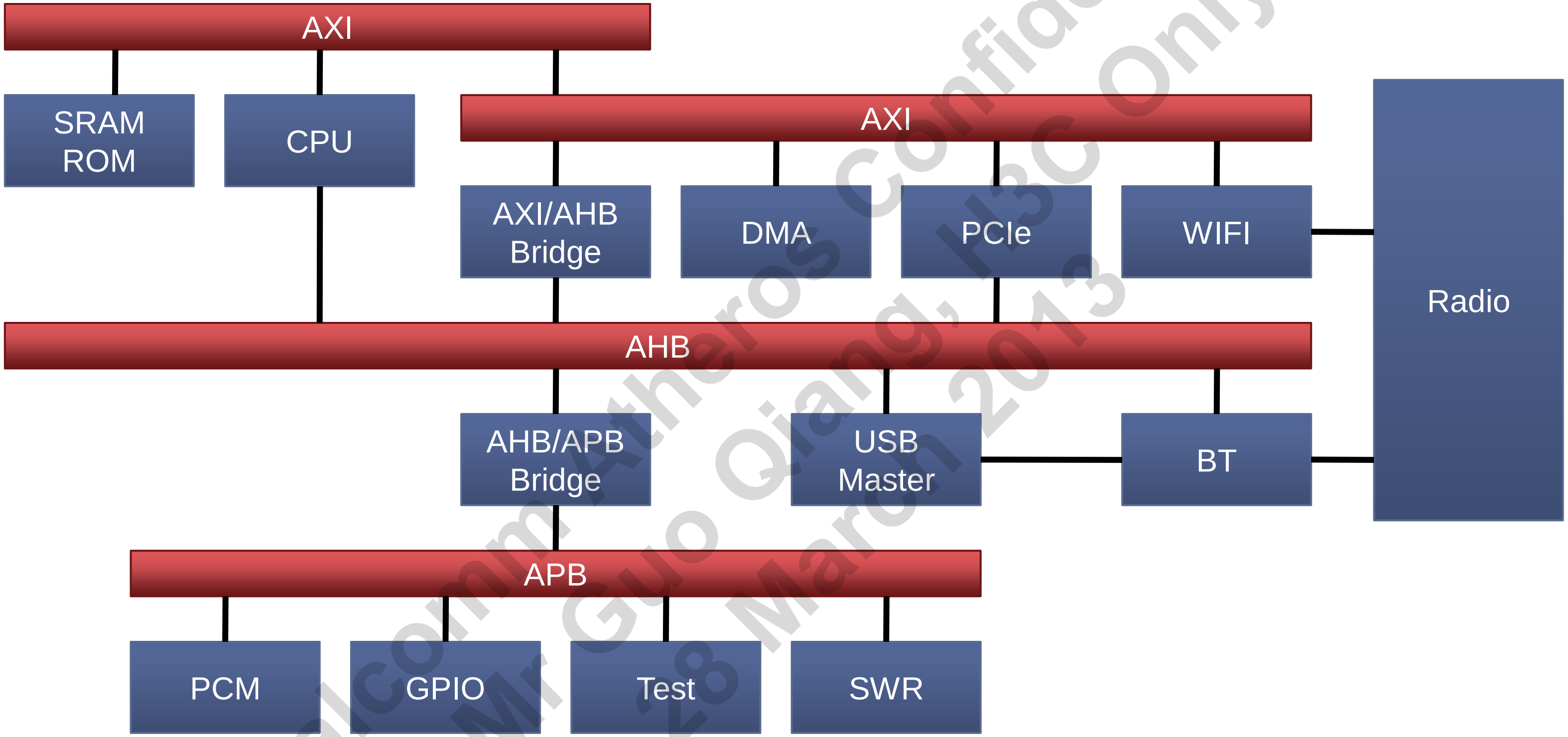
- Wider BW (80/160 MHz) in 11ac will increase the probability of hidden nodes on secondary channels

- VHT RTS/CTS
  - CTS is sent only on channels sensed to be idle by the responder
  - Initiator transmits data only over channels indicated free by CTS response



1.2 System Overview

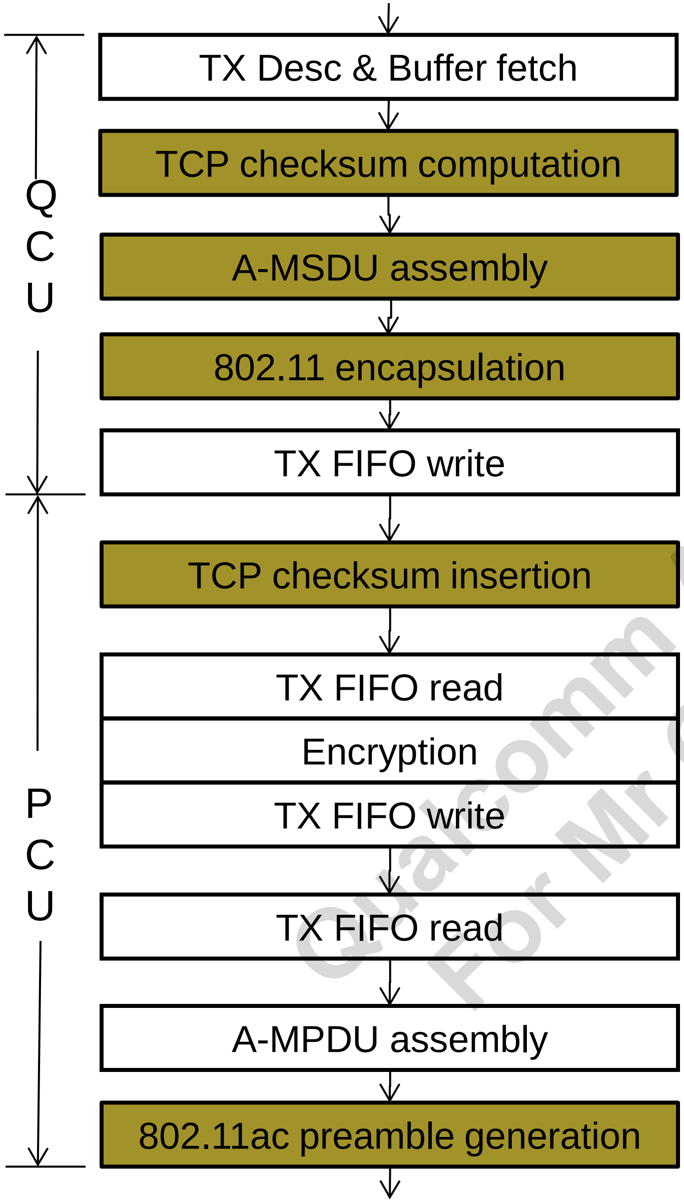
1.2.1 Peregrine Hardware



- Tensilica CPU @ 240 MHz
- Single cycle access to ROM and RAM
- One thread of execution

- Tight loop using a run-to-completion model
- > 2x code density compared to MIPS

11ac MAC Tx Data Flow

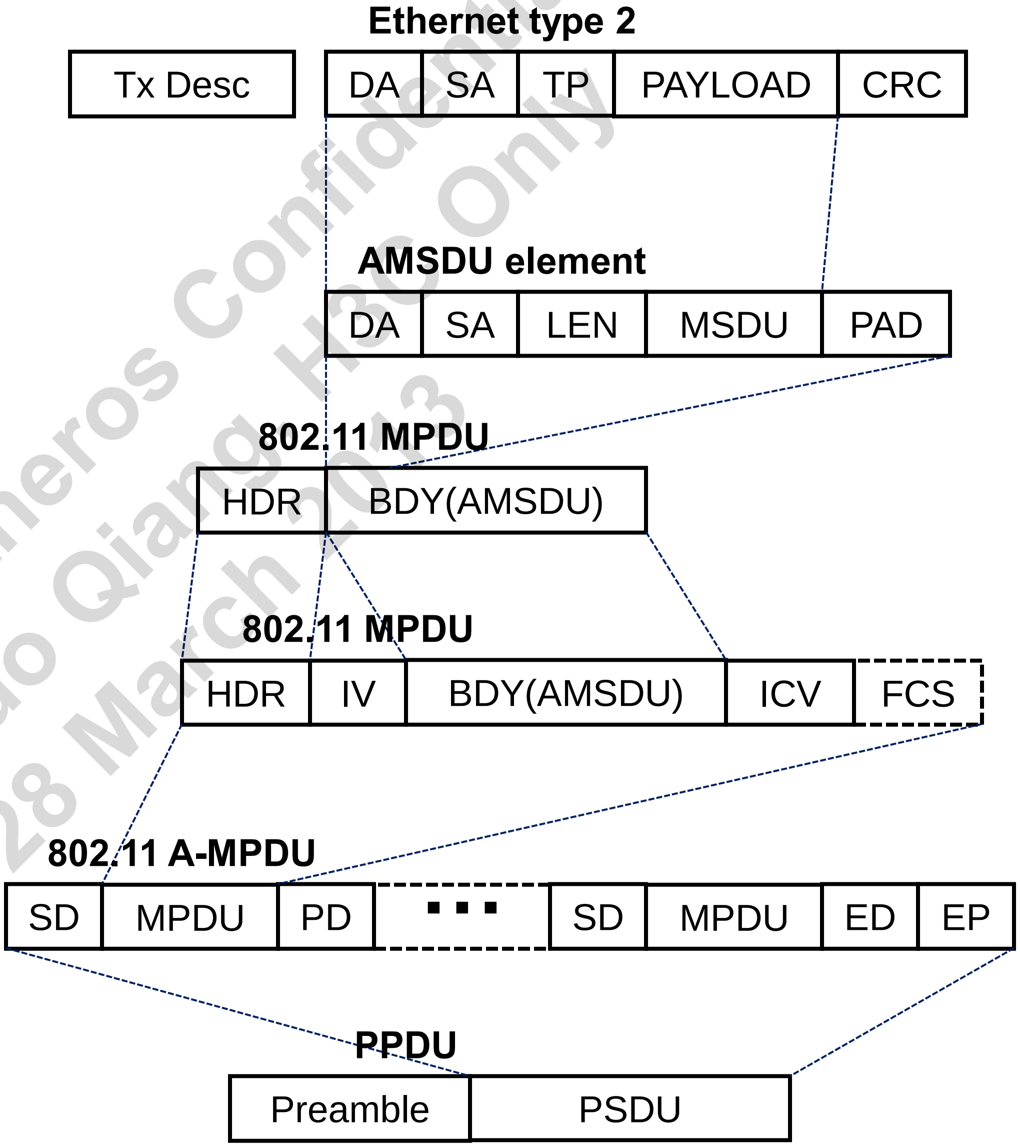


QCU

- TX Desc & Buffer fetch
- TCP checksum computation
- A-MSDU assembly
- 802.11 encapsulation
- TX FIFO write

PCU

- TCP checksum insertion
- TX FIFO read
- Encryption
- TX FIFO write
- TX FIFO read
- A-MPDU assembly
- 802.11ac preamble generation



Tx Desc

Ethernet type 2

DA | SA | TP | PAYLOAD | CRC

AMSDU element

DA | SA | LEN | MSDU | PAD

802.11 MPDU

HDR | BDY(AMSDU)

802.11 MPDU

HDR | IV | BDY(AMSDU) | ICV | FCS

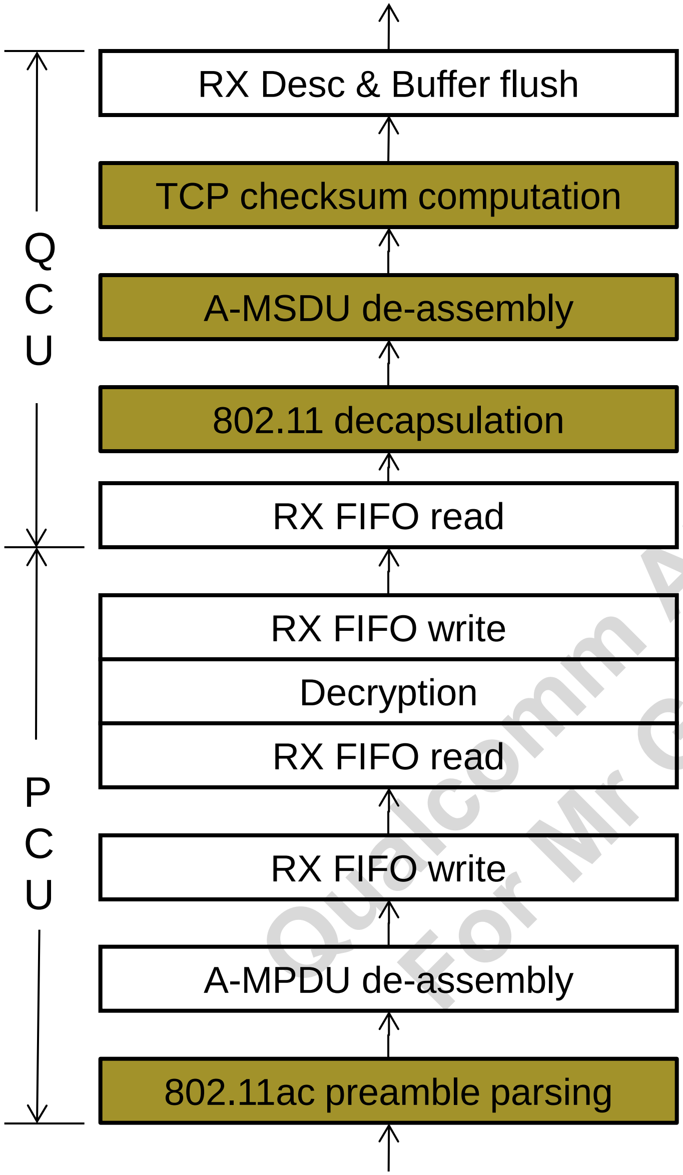
802.11 A-MPDU

SD | MPDU | PD | ▪ ▪ ▪ | SD | MPDU | ED | EP

PPDU

Preamble | PSDU

11ac MAC Rx Data Flow



QCU

RX Desc & Buffer flush

TCP checksum computation

A-MSDU de-assembly

802.11 decapsulation

RX FIFO read

PCU

RX FIFO write

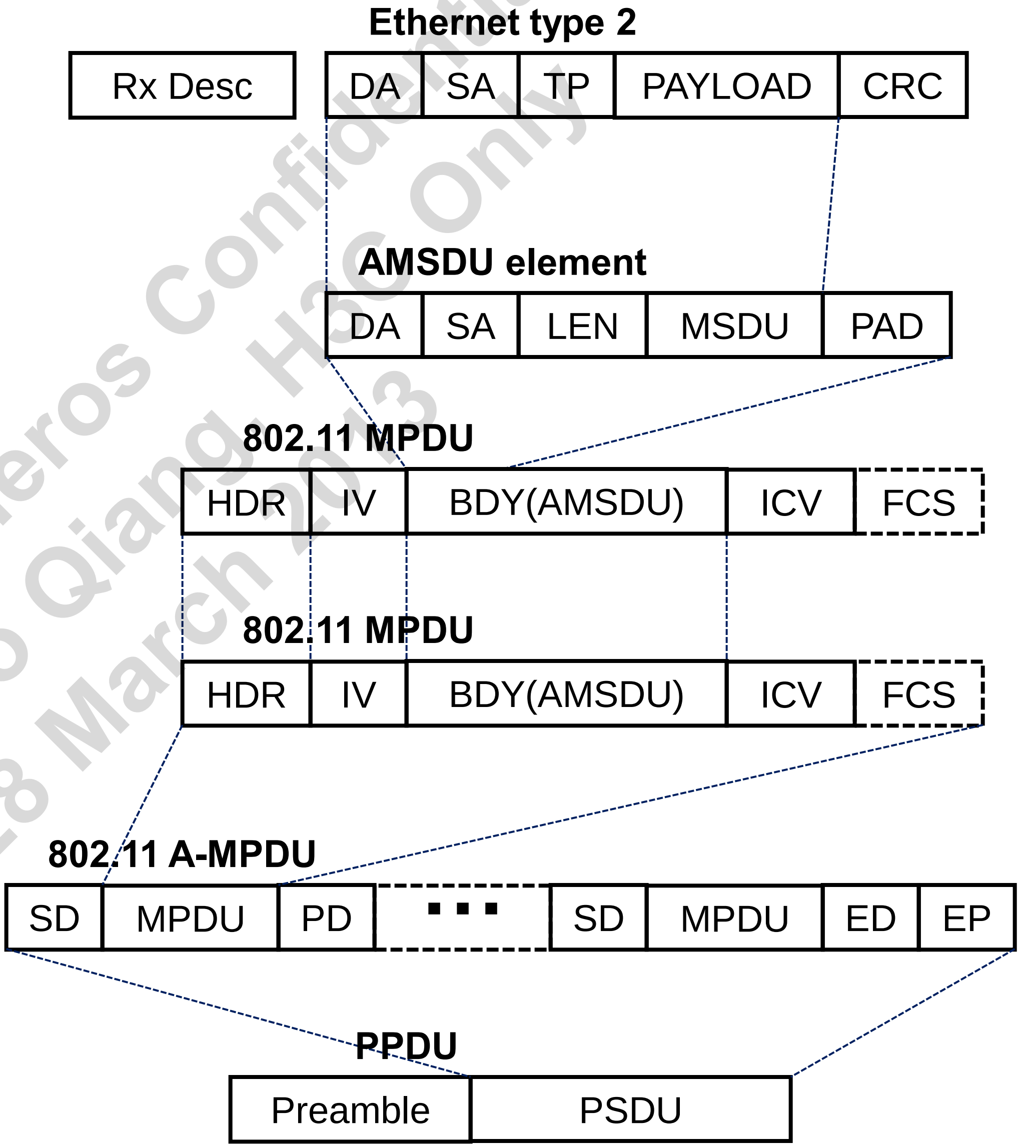
Decryption

RX FIFO read

RX FIFO write

A-MPDU de-assembly

802.11ac preamble parsing



Rx Desc

Ethernet type 2

DA | SA | TP | PAYLOAD | CRC

AMSDU element

DA | SA | LEN | MSDU | PAD

802.11 MPDU

HDR | IV | BDY(AMSDU) | ICV | FCS

802.11 MPDU

HDR | IV | BDY(AMSDU) | ICV | FCS

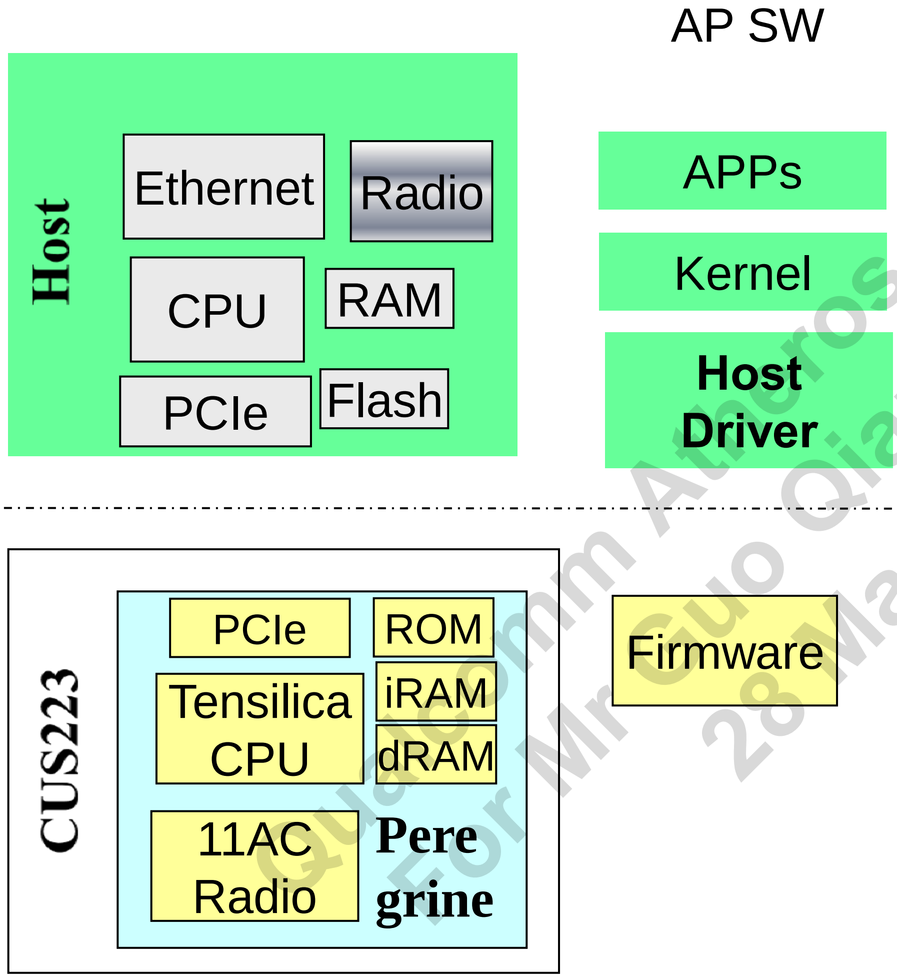
802.11 A-MPDU

SD | MPDU | PD | ... | SD | MPDU | ED | EP

PPDU

Preamble | PSDU

1.2.2 AP software



SW Releases

AP s/w Release

AP LSDK 10.1.45 Beta BSP Source Release [1/21/2013]

AP LSDK 10.1.45 Beta WLAN Driver Source Release [1/21/2013]

AP LSDK 10.1.45 Beta Spectral Analysis Source Release [1/21/2013]

AP LSDK 10.1.45 Beta Linux Host Offload Stack Source Release [1/21/2013]

AP LSDK 10.1.45 Beta Linux Target Offload Stack Source Release [1/21/2013]

CUS223 Firmware Release: ONLY available for customers with special licensing agreement

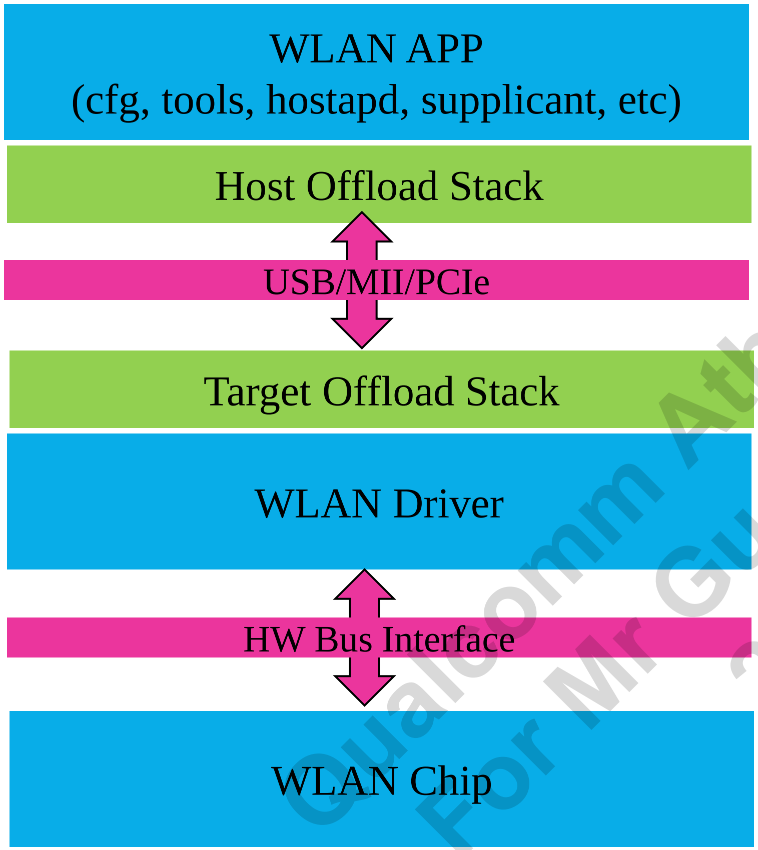
AP LSDK 10.1.45 Beta F/W Source Release [1/21/2013]

CUS223 Specific Firmware Toolchain Package

CUS223 Firmware Configuration Parameters

Software Offload Architecture

Full Offload



WLAN APP
(cfg, tools, hostapd, supplicant, etc)

Host Offload Stack

USB/MII/PCIe

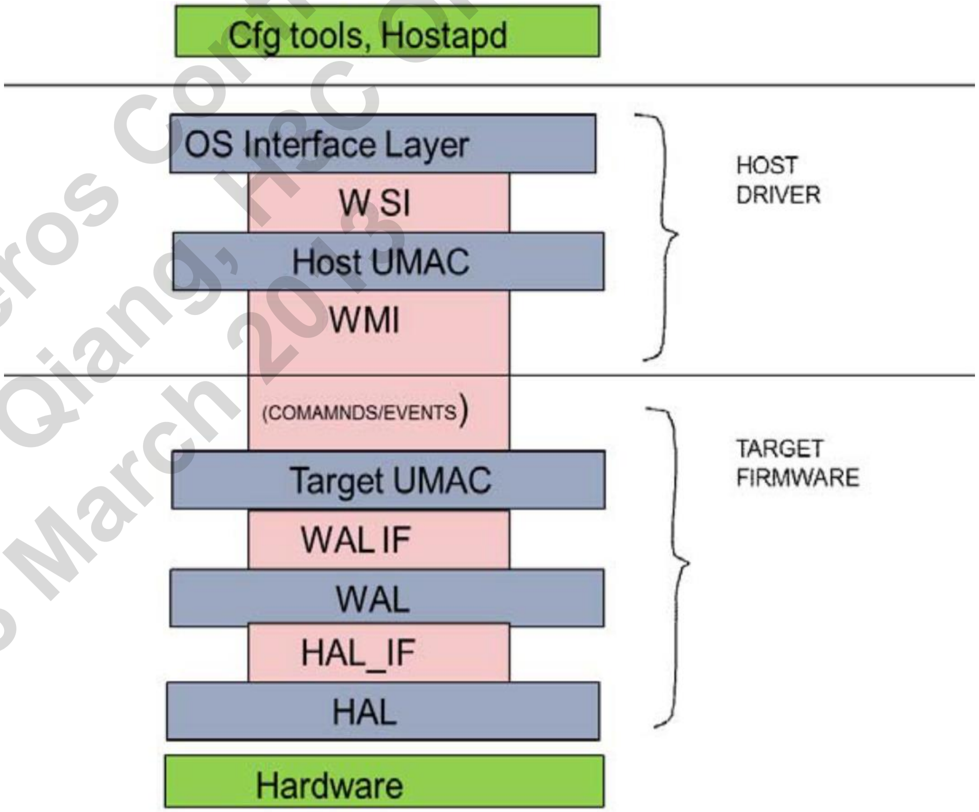
Target Offload Stack

WLAN Driver

HW Bus Interface

WLAN Chip

Target (Partial) Offload



Cfg tools, Hostapd

OS Interface Layer

W SI

Host UMAC

WMI

(COMAMNDS/EVENTS)

Target UMAC

WAL IF

WAL

HAL_IF

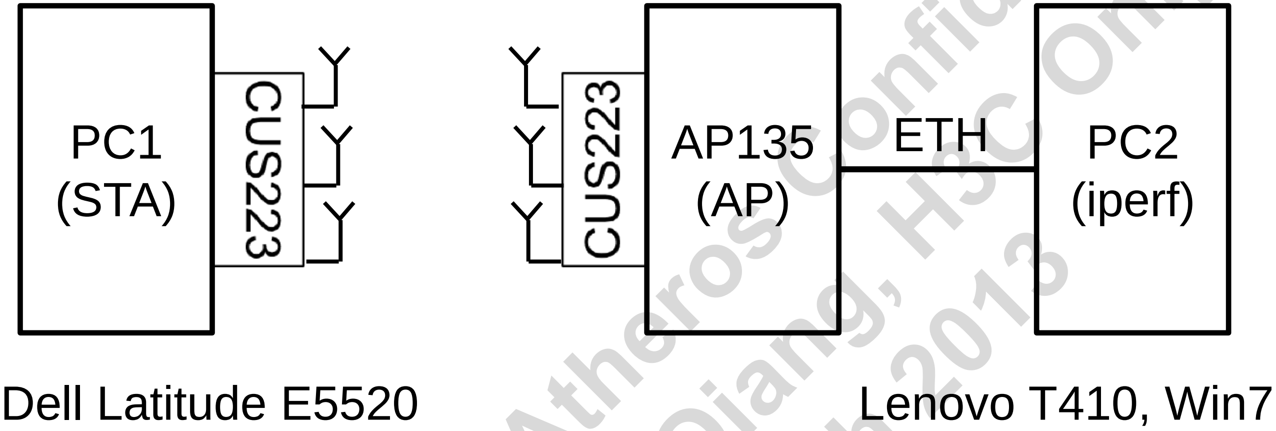
HAL

Hardware

HOST
DRIVER

TARGET
FIRMWARE

Test Setup



PC1
(STA)

CUS223

CUS223

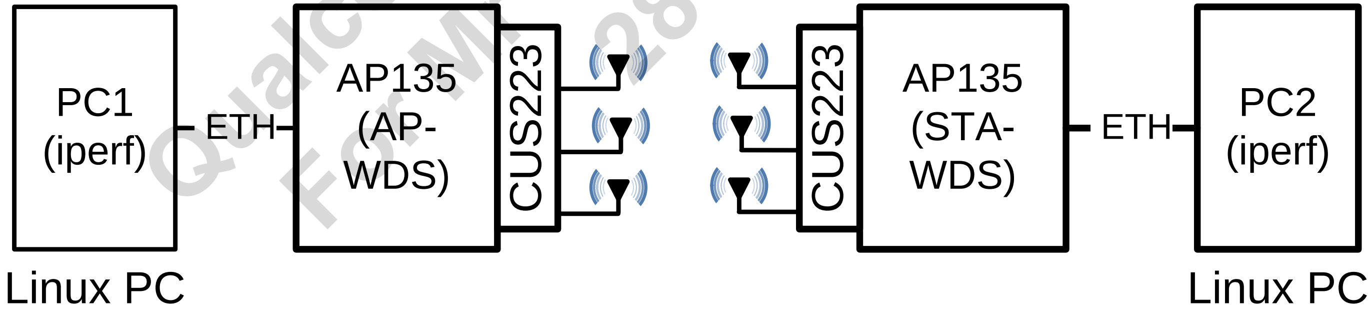
AP135
(AP)

ETH

PC2
(iperf)

Dell Latitude E5520

Lenovo T410, Win7



PC1
(iperf)

ETH

AP135
(AP-
WDS)

CUS223

CUS223

AP135
(STA-
WDS)

ETH

PC2
(iperf)

Linux PC

Linux PC

1.3 Release Documents

AP135/CUS223 Pre-RC Release Documents

AP135/CUS223 Pre-RC Customer Release Notes [12/24/2012]
AP135/CUS223 Pre-RC Setup Guide [12/24/2012]
AP135 Pre-RC Hardware Reference Guide [12/24/2012]
CUS223-021 DVT Report [01/14/2013] ?
CUS223-021 Hardware Reference Guide [12/24/2012]
AP135/CUS223 Pre-RC Firmware Build Guide [12/24/2012]
AP135/CUS223 ART2 Pre-RC Release Notes [12/24/2012]

Peregrine 3rd Party Porting Releases

WLAN Driver Build Guide APM Cascade App Note [12/24/2012]
WLAN Driver Build Guide Freescale P1020 App Note [12/24/2012]
WLAN Driver Build Guide Intel Puma 6 App Note [12/24/2012]
WLAN Driver Build Guide Ikanos Vx185 App Note [12/24/2012]
WLAN Driver Build Guide Lantiq_VRX288 App Note [12/24/2012]
WLAN Driver Build Guide Mindspeed C2K App Note [12/24/2012]

Freescale s/w Porting Release

Freescale Porting s/w 10.1.45 Release(Peregrige 2.0 Beta) [01/29/2013]
Patch for Freescale's Platform(10.1.45)[01/29/2013]
Throughput test result(10.1.45) [01/29/2013]

2. How to Build

2.1 FW Build

Build Procedure

```
===============
1) Change directory to perf_pwr_offloads/drivers
2) Set Xtensa related build environment variables
3) Set Firmware build environment variables
4) make -C target clobber (Clean)
5) make -C target
```

```
cd perf_pwr_offload/drivers/
export ATH_BUILD_TARG_LIST="AR9888"
./support/makesdk
```

The **env file** is as the following

```
XTENSA_CORE=peregrine
LM_LICENSE_FILE=`/cad/local/bin/flexlm -w tensilica`
XTENSA_PREFER_LICENSE=XT-GENERIC
XTENSA_TOOLS_ROOT=/cad/tensilica/tools/RD-2011.2-linux/XtensaTools
XTENSA_ROOT=/cad/tensilica/chips/peregrine/RD-2011.2-linux/peregrine
XTENSA_SYSTEM=$XTENSA_ROOT/config
PATH=$XTENSA_TOOLS_ROOT/bin:$PATH
export LM_LICENSE_FILE
export XTENSA_PREFER_LICENSE XTENSA
        _TOOLS_ROOT XTENSA_ROOT XTENSA
        _SYSTEM XTENSA_CORE
export PATH
```

Build Related Environment Variables

```
===========================
export WORKAREA=`pwd`
export TARGET=AR9888
export TARGET_VER=1
export TARGET_REV=1
export FPGA_FLAG=0
export FPGA_BB=1
```

Use prebuild FW

1) …/APQCAMain.156/firmware /Firmware.build_drivers.156.tgz

2) Unzip the file, and FW is under Firmware.build_drivers.156 /target/AR9888/hw1/bin/athwlan.bin

Install FW and Caldata

1) Install Caldata

```
sudo cp –rf qca_main_156/perf_pwr_offload/drivers/host/tools/systemtools/
tools/eepromUtil/qc98xx_template/boardData_1_1_QC98XX_cus223_gld.bin
/lib/firmware/fakeBoardData_AR6004.bin
```

2) Install FW

```
sudo cp athwlan.bin /lib/firmware/.
```

2.2 Host Driver Build

1) Untar the driver

```
# mkdir db120_11ac
# cd db120_11ac
# tar -xzvf LSDK-WLAN-999.999.0.138.tgz
```

2) Set Building environment

```
# export TOOLCHAIN=gcc-4.3.3
# export TOOLPREFIX=mips-linux-uclibc-
# export TOOLARCH=build_mips/staging_dir/usr
# export KERNEL=mips-linux-2.6.31
# export KERNELVER=2.6.31
# export KERNELTARGET=vmlinux.bin
# export KERNELARCH=mips
# export TARGETARCH=mipsisa32-be-elf
# export MAKE=make
# export MAKEARCH=${MAKE} ARCH=${KERNELARCH} CROSS_COMPILE=${TOOLPREFIX}
# export TOPDIR=db120_11ac
# export TOOLPATH=${TOPDIR}/build/${TOOLCHAIN}/${TOOLARCH}/
# export PATH=${TOPDIR}/build/util:${TOOLPATH}/bin:
${TOPDIR}/linux:${TOPDIR}/build:`pwd`:${PATH}
# export KERNELPATH=${TOPDIR}/linux/kernels/${KERNEL}
# export BOARD_TYPE=<board type> (ex: db12x)
# export BUILD_EXT=<ext if any > (ex: _s17)
# export INSTALL_ROOT=${TOPDIR}/rootfs-${BOARD_TYPE}
${BUILD_CONFIG}${BUILD_EXT}${NAND}.build
# export MODULEPATH=${INSTALL_ROOT}/lib/modules/${KERNELVER}/net
# export WIRELESSTOOLS=wireless_tools.29
# export WIRELESSTOOLSLIB=libiw.so.29
# export HAL=${TOPDIR}/drivers/wlan_modules/hal
# export ATHEROSPATH=${TOPDIR}/drivers/wlan_modules
# export ATH_PERF_PWR_OFFLOAD=1
# export ATH_TGT_TYPE=AR9888
# export ATH_HIF_TYPE=pci
# export FORCE_LEGACY_PCI_INTERRUPTS=1
# export LOAD_ARRAY_FW=1
# export BIG_ENDIAN_HOST=1
# export REMOVE_PKT_LOG=1
```

3) Building HAL

```
# cd ${HAL}/linux
# make TARGET=${TARGETARCH} clean
# make TARGET=${TARGETARCH}
# make TARGET=${TARGETARCH} release
```

4) Building WLAN Driver

```
# cd ${TOPDIR}/drivers/wlan_modules/os/linux
# ${MAKEARC} TARGET=${TARGETARCH} clean
# ${MAKEARCH} TARGET=${TARGETARCH}
# ${MAKEARCH} DESTDIR=${INSTALLROOT} TARGET=${TARGETARCH} install
```

5) Building Wireless Tools

Buidling wlanconfig:

```
# cd ${TOPDIR}/drivers/wlan_modules/os/linux/tools
# make clean
# make wlanconfig BUILD_STATIC=${BUILD_STATIC}
# cp -f wlanconfig ${INSTALL_ROOT}/sbin
```

Buidling ath-tools:

```
# cd ${TOPDIR}/drivers/wlan_modules/os/linux/tools
# make ath_tools_clean;
# make ath_tools
```

6) Building hostap Applications

```
# cp -f scripts/${BOARD_TYPE}/athr_hostapd.conf ../apps/athr-
hostap/hostapd/.config
# cd {TOPDIR}/apps/athr-hostap/hostapd
# make clean
# make CC=${TOOLPREFIX}gcc AR=${TOOLPREFIX}ar LD=${TOOLPREFIX}ld
# cp hostapd hostapd_cli ${INSTALL_ROOT}/sbin
# cp {TOPDIR}/rootfs/cgiCommon/etc/ath/hostapd0.7.0_conf/*
${INSTALL_ROOT}/etc/ath
# rm -rf ${INSTALL_ROOT}/etc/ath/hostapd0.7.0_conf
# mkdir -p ${INSTALL_ROOT}/etc/wpa2
```

7) Building the athr-wpa_supplicant:

```
# cp -f scripts/${BOARD_TYPE}/athr_supplicant.conf
  {TOPDIR}/apps/athr-hostap/wpa_supplicant/.config
# cd {TOPDIR}/apps/athr-hostap/wpa_supplicant
# make clean
# make CC=${TOOLPREFIX}gcc AR=${TOOLPREFIX}ar LD=${TOOLPREFIX}ld;
# cp wpa_supplicant wpa_cli ${INSTALL_ROOT}/sbin;
```

Script Build

1) X86 Host Driver

```
mkdir qca_main_156
cd qca_main_156
tar -zxvf ~/share/qca_main_156/LSDK-X86HOST-999.999.0.168.tgz
cd build
make BOARD_TYPE=x86-host-small
```

2) AP135 Host Driver

```
# mkdir <path-to-home-diredtory>/ap135_11ac
# cd ap135_11ac
# tar –xzvf LSDK-999.999.0.238.tgz
# tar –xzvf LSDK-WLAN-999.999.0.238.tgz
# cd build
# make BOARD_TYPE=ap135 11AC_OFFLOAD=1
```

3. AP/STA/ART2 Operation

3.1 Configuration Parameters

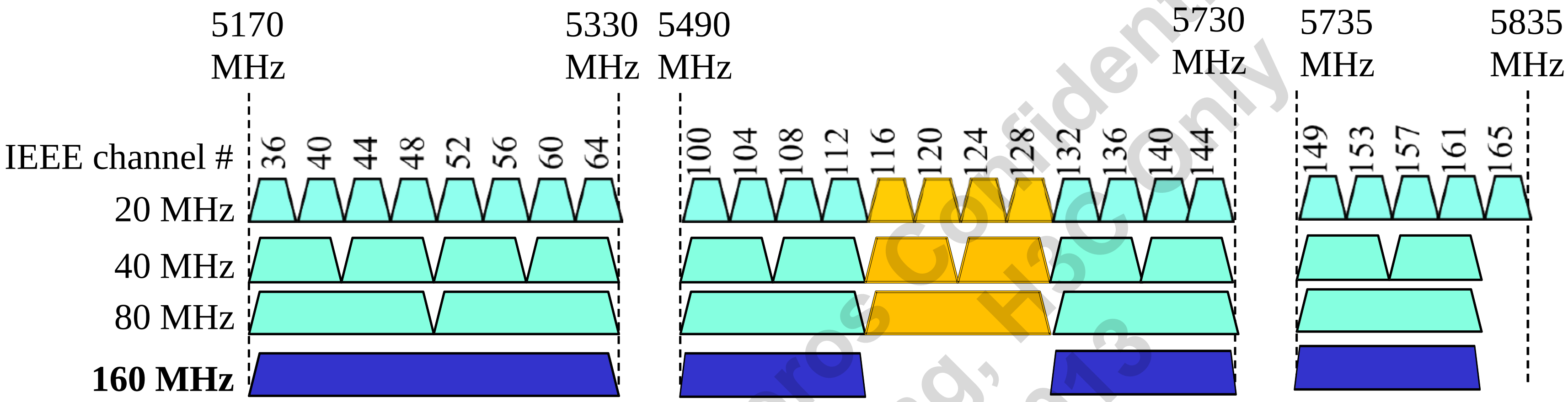
3.1.1 Channel Selection

Channel Usage before 11ac

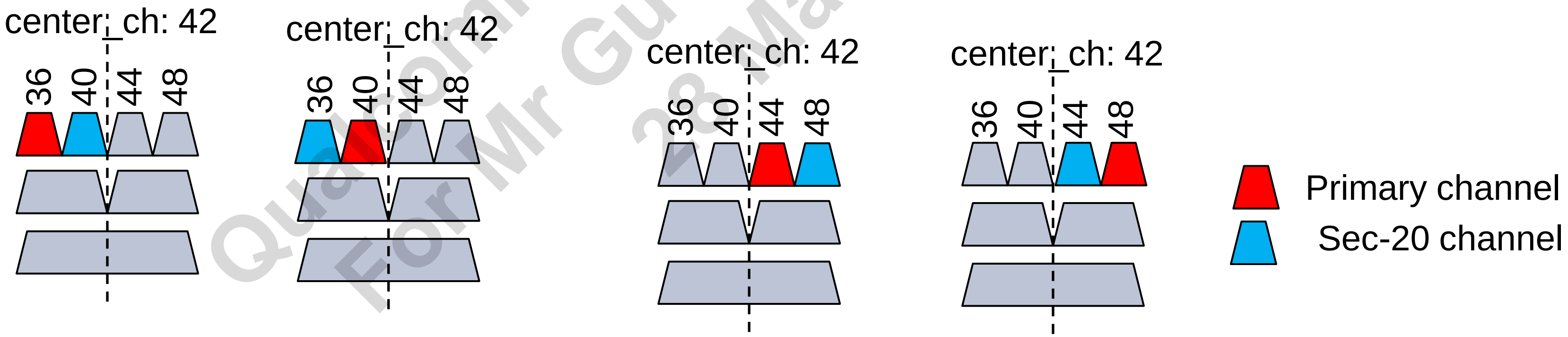
chn	Freq						
	(Ghz)				legacy	HT20	HT40
1	2.412	🟥	🟨	🟩	11b/g	HT20	Ch1+ Ch5
2	2.417	🟥	🟨	🟩	11b/g	HT20	Ch2+ Ch6
3	2.422	🟥	🟨	🟩	11b/g	HT20	Ch3+ Ch7
4	2.427	🟥	🟨	🟩	11b/g	HT20	Ch4+ Ch8
5	2.432	🟥	🟨	🟩	11b/g	HT20	Ch5+ Ch9
6	2.437	US	EU	JP	11b/g	HT20	Ch6+ Ch10
7	2.442	🟥	🟨	🟩	11b/g	HT20	Ch7+ Ch11
8	2.447	🟥	🟨	🟩	11b/g	HT20	Ch8+ Ch12
9	2.452	🟥	🟨	🟩	11b/g	HT20	Ch9+ Ch13
10	2.457	🟥	🟨	🟩	11b/g	HT20	Ch10+ Ch6
11	2.462	🟥	🟨	🟩	11b/g	HT20	Ch11+ Ch7
12	2.467		🟨	🟩	11b/g	HT20	Ch12+ Ch8
13	2.472		🟨	🟩	11b/g	HT20	Ch13+ Ch9
14	2.484			🟩	11b		
34	5.170			JP	11a		
38	5.190			🟩	11a		
42	5.210			🟩	11a		
46	5.230			🟩	11a		

36	5.180	US	EU	JP	11a	HT20	HT40	VHT 80
40	5.200	🟥	🟨	🟩	11a	HT20		
44	5.220	🟥	🟨	🟩	11a	HT20	HT40	
48	5.240	🟥	🟨	🟩	11a	HT20		
				🟩				
52	5.260	🟥	🟨	🟩	11a	HT20	HT40	VHT 80 DFS
56	5.280	🟥	🟨	🟩	11a	HT20		
60	5.300	🟥	🟨	🟩	11a	HT20	HT40	
64	5.320	🟥	🟨	🟩	11a	HT20		
				🟩				
100	5.500	🟥	🟨	🟩	11a	HT20	HT40	VHT 80 DFS
104	5.520	🟥	🟨	🟩	11a	HT20		
108	5.540	🟥	🟨	🟩	11a	HT20	HT40	
112	5.560	🟥	🟨	🟩	11a	HT20		
116	5.580	🟥	🟨	🟩	11a	HT20	HT40	VHT 80 DFS
120	5.600	🟥	🟨	🟩	11a	HT20		
124	5.620	🟥	🟨	🟩	11a	HT20	HT40	
128	5.640	🟥	🟨	🟩	11a	HT20		
132	5.660	🟥	🟨	🟩	11a	HT20	HT40	
136	5.680	🟥	🟨	🟩	11a	HT20		
140	5.700	🟥	🟨	🟩	11a	HT20		
149	5.745	🟥			11a	HT20	HT40	VHT 80
153	5.765	🟥			11a	HT20		
157	5.785	🟥			11a	HT20	HT40	
161	5.805	🟥			11a	HT20		
165	5.825	🟥			11a	HT20		

11ac Channelization



Note: the 80 MHz channel spanning Channels 116 to 128 is NOT allowed at this time due to TDWR restrictions. So, we don't use it (show it in Yellow).



3.1.2 PHY Rate

11n PHY Rate – determined by MCS

MCS index	Nss	Rate (GI = 800ns)		Rate (GI = 400ns)	
		20MHz	40MHz	20MHz	40MHz
0	1	6.50	13.50	7.22	15.00
1	1	13.00	27.00	14.44	30.00
2	1	19.50	40.50	21.67	45.00
3	1	26.00	54.00	28.89	60.00
4	1	39.00	81.00	43.33	90.00
5	1	52.00	108.00	57.78	120.00
6	1	58.50	121.50	65.00	135.00
7	1	65.00	135.00	72.22	150.00
8	2	13.00	27.00	14.44	30.00
9	2	26.00	54.00	28.89	60.00
10	2	39.00	81.00	43.33	90.00
11	2	52.00	108.00	57.78	120.00
12	2	78.00	162.00	86.67	180.00
13	2	104.52	216.00	116.13	240.00
14	2	117.00	243.00	130.00	270.00
15	2	130.00	270.00	144.44	300.00

MCS Index	Modulation	R	Data rate (Mbps)	
			NGI	SGI
16	BPSK	½	40.5	45.0
17	QPSK	½	81.0	90.0
18	QPSK	¾	121.5	135.0
19	16-QAM	½	162.0	180.0
20	16-QAM	3/4	243.0	270.0
21	64-QAM	2/3	324.0	360.0
22	64-QAM	¾	364.5	405.0
23	64-QAM	5/6	405.0	450.0

Mode	Rate (Mb/s)	Mode	Rate (Mb/s)
b	1	g	12
	2		18
	5.5		24
	11		36
g	6		48
	9		54

11ac PHY Rate

11ac PHY rate depends on the following factors.

- MCS number (called vthmcs)
- Channel bandwidth (20/40/80MHz)
- Number of streams (Nss =1/2/3)
- You have to specify chainmask correctly. Otherwise, you can’t get the required Nss.
- Guard Interval (Full/Short GI)

MCS		Nss=1		Nss=3	
No	**Modulation**	40 MHz	80 MHz	40 MHz	80 MHz
0	BPSK 1/2	15.0	32.5	45.0	97.5
1	QPSK 1/2	30.0	65.0	90.0	195.0
2	QPSK 3/4	45.0	97.5	135.0	292.5
3	16-QAM 1/2	60.0	130.0	180.0	390.0
4	16-QAM 3/4	90.0	195.0	270.0	585.0
5	64-QAM 2/3	120.0	260.0	360.0	780.0
6	64-QAM 3/4	135.0	292.5	405.0	-
7	64-QAM 5/6	**150.0**	325.0	**450.0**	975.0
8	256-QAM 3/4	180.0	390.0	540.0	1170.0
9	256-QAM 5/6	200.0	433.3	600.0	1300.0

- Data rate assuming short GI
- Nss: Number of spatial streams
- Some BW+MCS combinations are not allowed to simplify TX flow
  - ✓Nss=3 with 64-QAM 3/4, Nss=3 with 256-QAM 5/6

You have to specify all the relevant parameters correctly to get the required PHY rate.

- iwpriv athN vhtmcs <mcsindex>

This command specifies the VHT MCS Index to be used with data frame transmissions. Note that invoking this command with valid MCS Index (0-9) enables "fixed rate" and Invalid index disables fixed rate setting.

- iwpriv athN nss <spatial_streams>

This command specifies the number of Spatial Streams, 1~3, to be enabled.

- iwpriv athN tx(rx)_chainmask mask

These parameters set the transmit and receive chainmask values. These setting affect ALL VAPS, not just the VAP that is being set. The default chainmask values are stored in EEPROM. This iwpriv command will override the current chainmask settings.

- iwpriv athN chwidth ChannelWidth

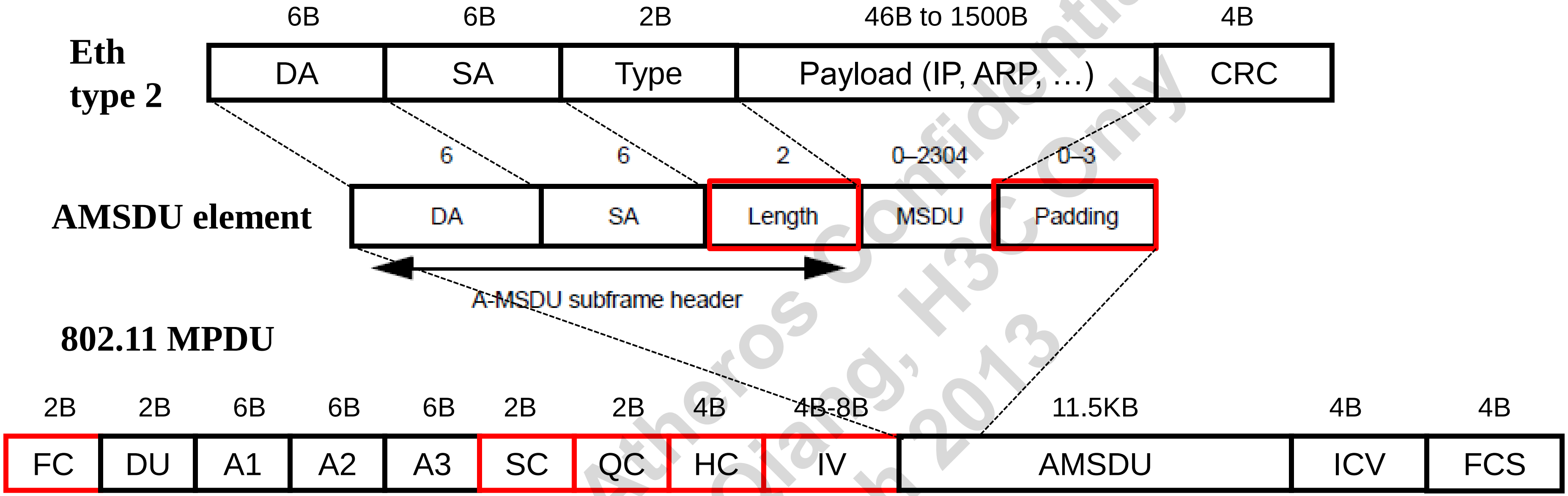
This command sets the Channel Width field in the AP beacons High Throughput Information Element (HT IE) or Very High Throughput IE when applicable. 0 - Use the device settings, 1 - 20 MHz, 2 - 20/40 MHz, 3 - 20/40/80 MHz

- iwpriv athN shortgi 1|0

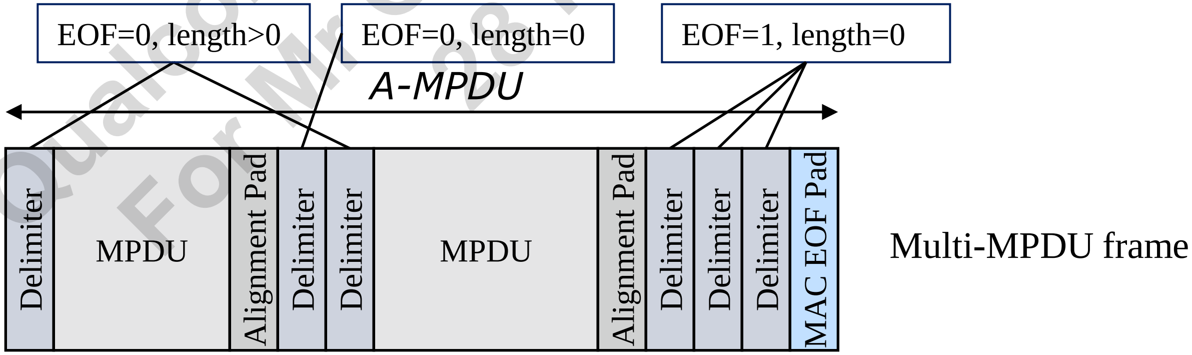
This command will enable/disable the short Gating Interval (shortgi). This effectively increases the PHY rate by 25%. Its default value is 1.

3.1.3 Aggregation

AMSDU



AMPDU



Aggregation configurations

- iwpriv athN amsdu isEnable → enable/disable transmission of AMSDU
Reception of ASMDU is supported by default. The default setting is 1 (enabled).
- iwpriv athN amsdulimit → Set the number of subframes in an A-MSDU.
- iwpriv wifiN AMPDU 1|0 → Enable/disable transmission of AMPDU
Receiving of aggregate frames will still be performed, but no aggregate frames will be transmitted if this is disabled. The default value is 1 (enabled).
- iwpriv athN ampdulimit → limits the number of bytes included in an AMPDU.
Frames add to an aggregate until either the transmit duration is exceeded, the number of subframes is exceeded, the maximum number of bytes is exceeded, or the corresponding queue is empty. The default value is 50kB.
-iwpriv athN ampdusframes numFrames → set the maximum number of subframes to place into an AMPDU. The default value is 32.

We want to squeaze more frames in a amsdu/ampdu. But, there are limits.
For **amsdu**,

```
if (max_subfrms_amsdu && (max_subfrms_amsdu < 32)) {
HTT_AGGR_CFG_MAX_NUM_AMSDU_SUBFRM_SET(*msg_word,
max_subfrms_amsdu);}
```

For **ampdu**,

```
if (max_subfrms_ampdu && (max_subfrms_ampdu <= 64)) {
HTT_AGGR_CFG_MAX_NUM_AMPDU_SUBFRM_SET(*msg_word,
max_subfrms_ampdu); }
```

3.1.4 Advanced FEC

LDPC (Low Density Parity Check codes)

- A high performance error correction code -- close to Shannon limit
- Linear block code
- 802.11 defines three codeword lengths: 648, 1296 and 1944, with 4 code rates (1/2, 2/3, 3/4 and 5/6) for each of them
- Decoder is computational intensive
  - Standard algorithm is called belief propagation algorithm
  - Iterative decoding
- Simulations show ~2dB performance gain over convolutional code

- iwpriv athN ldpc <0|1>

This command allows enabling/disabling of LDPC.
0 - Disable LDPC,
1 - Enable LDPC

3.2 Configuration Options

3.2.1 Direct Configuration

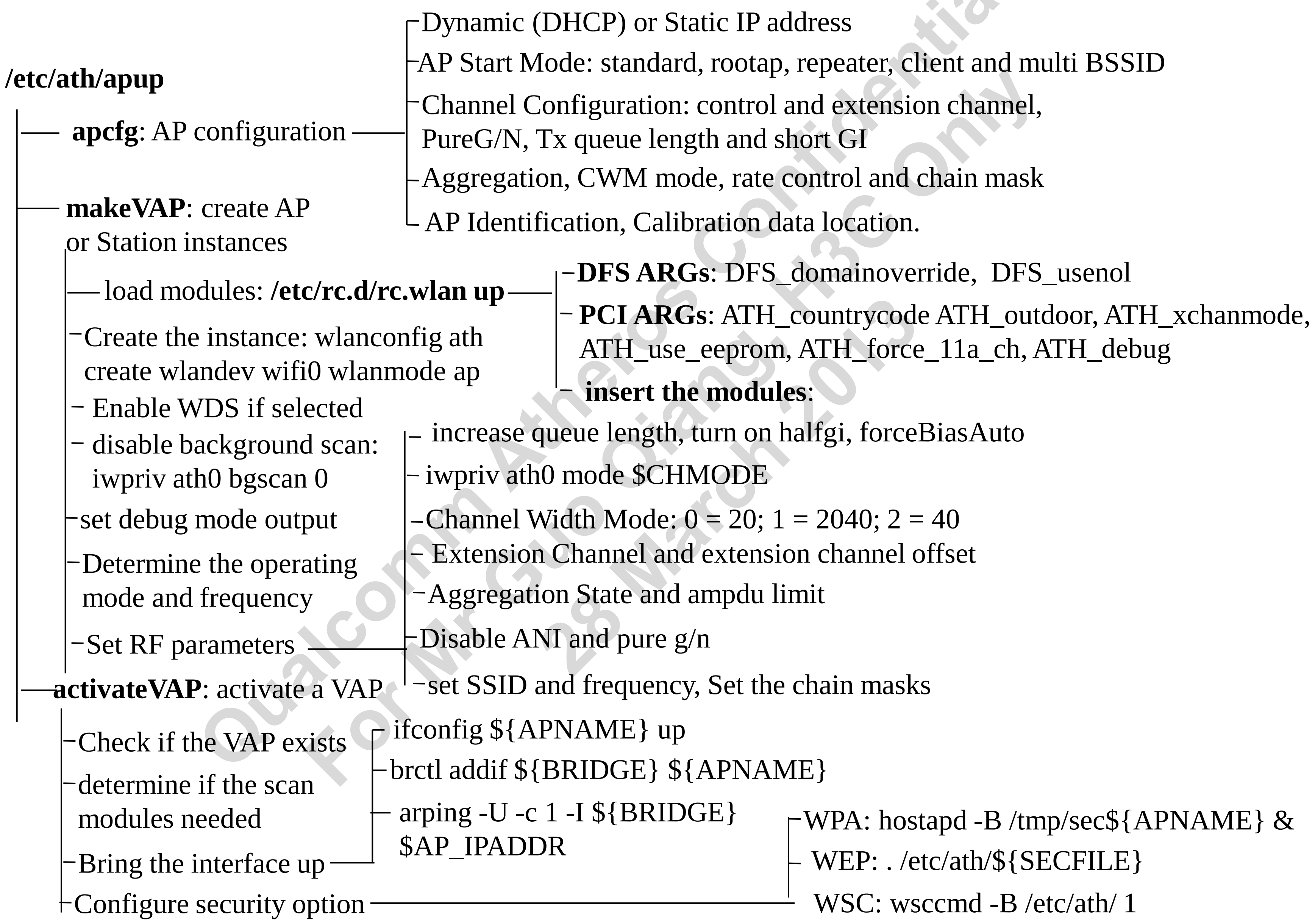
A 11n (newma) AP configuration

```
export AP_PRIMARY_CH=36
export MODULE_PATH=/home/george/ath_pcap/rootfs.build/lib/modules/2.6.22-14-generic/net
/sbin/insmod $MODULE_PATH/ath_hal.ko
/sbin/insmod $MODULE_PATH/wlan.ko
/sbin/insmod $MODULE_PATH/ath_rate_atheros.ko
/sbin/insmod $MODULE_PATH/ath_dfs.ko
/sbin/insmod $MODULE_PATH/ath_dev.ko
/sbin/insmod $MODULE_PATH/ath_pci.ko
/sbin/insmod $MODULE_PATH/wlan_xauth.ko
/sbin/insmod $MODULE_PATH/wlan_ccmp.ko
/sbin/insmod $MODULE_PATH/wlan_tkip.ko
/sbin/insmod $MODULE_PATH/wlan_wep.ko
/sbin/insmod $MODULE_PATH/wlan_acl.ko
```

```
wlanconfig ath create wlandev wifi0 wlanmode ap

iwpriv ath0 bgscan 0
ifconfig wifi0 txqueuelen 1000
ifconfig ath0 txqueuelen 1000
iwpriv ath0 mode 11NAHT40PLUS
iwconfig ath0 channel ${AP_PRIMARY_CH}
iwpriv ath0 cwmmode 1
iwpriv ath0 extoffset 1
iwpriv ath0 extprotspac 0
iwpriv ath0 ampdu 1
iwpriv ath0 ampdulimit 50000
iwpriv ath0 tx_chainmask 3
iwpriv ath0 rx_chainmask 3
iwconfig ath0 essid "gml_tst_ap" mode master
ifconfig ath0 192.168.2.20 up
```

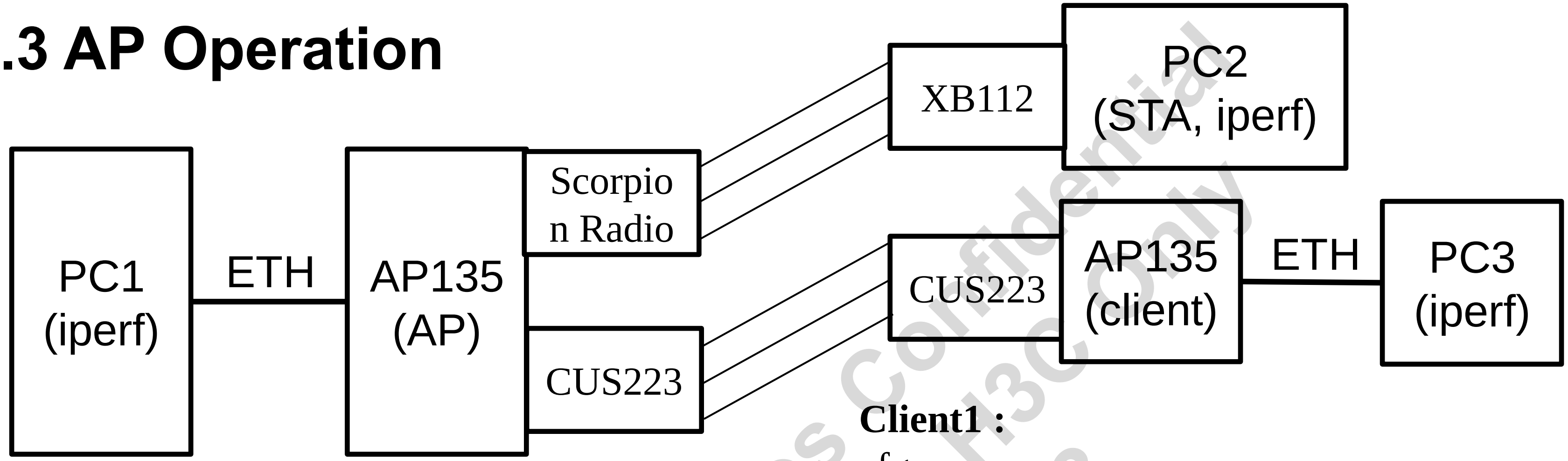
3.2.2 Script Configuration



/etc/ath/apup

- **apcfg**: AP configuration
  - Dynamic (DHCP) or Static IP address
  - AP Start Mode: standard, rootap, repeater, client and multi BSSID
  - Channel Configuration: control and extension channel, PureG/N, Tx queue length and short GI
  - Aggregation, CWM mode, rate control and chain mask
  - AP Identification, Calibration data location.
- **makeVAP**: create AP or Station instances
  - load modules: **/etc/rc.d/rc.wlan up**
    - **DFS ARGs**: DFS_domainoverride, DFS_usenol
    - **PCI ARGs**: ATH_countrycode ATH_outdoor, ATH_xchanmode, ATH_use_eeprom, ATH_force_11a_ch, ATH_debug
    - **insert the modules**:
  - Create the instance: wlanconfig ath create wlandev wifi0 wlanmode ap
  - Enable WDS if selected
  - disable background scan: iwpriv ath0 bgscan 0
  - set debug mode output
  - Determine the operating mode and frequency
  - Set RF parameters
    - increase queue length, turn on halfgi, forceBiasAuto
    - iwpriv ath0 mode $CHMODE
    - Channel Width Mode: 0 = 20; 1 = 2040; 2 = 40
    - Extension Channel and extension channel offset
    - Aggregation State and ampdu limit
    - Disable ANI and pure g/n
    - set SSID and frequency, Set the chain masks
- **activateVAP**: activate a VAP
  - Check if the VAP exists
  - determine if the scan modules needed
  - Bring the interface up
    - ifconfig ${APNAME} up
    - brctl addif ${BRIDGE} ${APNAME}
    - arping -U -c 1 -I ${BRIDGE} $AP_IPADDR
  - Configure security option
    - WPA: hostapd -B /tmp/sec${APNAME} &
    - WEP: . /etc/ath/${SECFILE}
    - WSC: wsccmd -B /etc/ath/ 1

3.3 AP Operation



AP:

```
cfg -x
cfg -a AP_IPADDR=192.168.1.2
cfg -a AP_STARTMODE=dual
cfg -a AP_SSID=ap135_11n
cfg -a AP_CHMODE_2=11ACVHT80
cfg -a AP_MODE_2=ap-wds
cfg -a AP_SSID_2=ap135_11ac
cfg -c
```

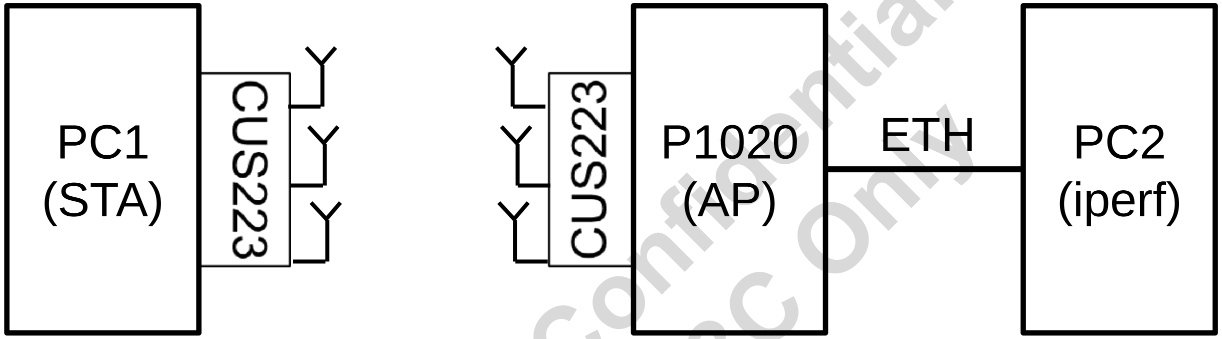
Client1 :

```
cfg -x
cfg -a AP_IPADDR=192.168.1.3
cfg -a AP_STARTMODE=client
cfg -a AP_RADIO_ID=1
cfg -a AP_CHMODE_2=11ACVHT80
cfg -a AP_SSID=ap135_11ac
cfg -a AP_MODE_2=sta-wds
cfg  -c
```

Or:

```
cfg -x
cfg -a AP_IPADDR=192.168.1.3
cfg -a AP_STARTMODE=dual
cfg -a AP_CHMODE_2=11ACVHT80
cfg -a AP_MODE_2=sta-wds
cfg -a AP_SSID_2=ptao_ap135_11ac
cfg -c
```

P1020 AP



```
[root@P1020RDB /root]# /etc/rc.d/rc.wlan up
[root@P1020RDB /root]# wlanconfig wlan create wlandev wifi0 wlanmode ap
[root@P1020RDB /root]# iwpriv wlan0 mode 11ACVHT80
[root@P1020RDB /root]# iwconfig wlan0 channel 48
[root@P1020RDB /root]# iwconfig wlan0 essid p1020_11ac
[root@P1020RDB /root]# iwpriv wlan0 chwidth 3
[root@P1020RDB /root]# iwpriv wlan0 nss 3
[root@P1020RDB /root]# iwpriv wlan0 shortgi 1
[root@P1020RDB /root]# iwpriv wlan0 ldpc 1
[root@P1020RDB /root]# iwpriv wlan0 vhtmcs 9
[root@P1020RDB /root]# iwpriv wlan0 amsdu 3
[root@P1020RDB /root]# iwpriv wlan0 ampdu 64
[root@P1020RDB /root]# brctl addif br-lan wlan0
[root@P1020RDB /root]# ifconfig wlan0 up
```

3.4 STA Operation

1) STA Software

The software provided for the customers to create 11ac client are as the following.

- Linux 3.1.0 Kernel.
  - ✓linux-headers-3.1.0-rc4-athos-ce-dev-wl_3.1.0-rc4-athos-ce-dev-wl-10.00.Custom_i386.deb
  - ✓linux-image-3.1.0-rc4-athos-ce-dev-wl_3.1.0-rc4-athos-ce-dev-wl-10.00.Custom_i386.deb
- 11ac driver and tools binaries.
  - ✓cus223_sw.tgz
- The above files can be downloaded from the support website.
  - ✓**CUS223 STA s/w Release**
  - ✓Linux Kernel Header
  - ✓Linux Kernel Image
  - ✓11ac driver and tools binaries

2) Preparing the Kernel

- Install Ubuntu 11.10 into your laptop, and use the following instructions to upgrade the kernel to 3.1.0.
- Install the new kernel

```
# dpkg -i linux-headers-3.1.0-rc4-athos-ce-dev-wl_3.1.0-rc4-athos-ce-dev-wl-10.00.Custom_i386.deb
# dpkg -i  linux-image-3.1.0-rc4-athos-ce-dev-wl_3.1.0-rc4-athos-ce-dev-wl-10.00.Custom_i386.deb
```

- Reboot to the new kernel

Installing 11ac Driver and Tools

- Please download cus223_sw.tgz from the support website, and use the following commands to untar and install the driver and tools.

```
# sudo su
# cd /root
# tar –zxvf cus223_sw.tgz
# cd cus223_sw
# ./cus223_ins.sh
```

Starting Up Client

Use the following command to start the client.

```
# ./11ac_up sta
```

11ac_up script

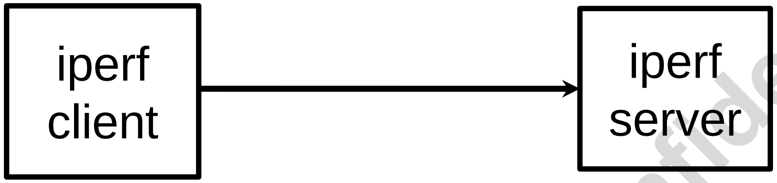
```
iwconfig wlan0 txpower off
ifconfig wlan0 down
rmmod iwlagn mac80211 cfg80211
echo 1 > /sys/bus/pci/rescan
sleep 1
rc.wlan up
sleep 4

wlanconfig wlan create wlandev wifi0 wlanmode sta
sleep 1
iwpriv wlan0 mode 11ACVHT80
iwpriv wlan0  chwidth 3
iwpriv wlan0  nss 3
iwpriv wlan0 ldpc 1
iwpriv wlan0 vhtmcs 9
iwpriv wlan0 amsdu 3
iwpriv wlan0 ampdu 64
iwconfig wlan0 essid p1020_11ac
ifconfig wlan0 192.168.1.99 up
```

Note:
You may have to remove the internal WiFi card before you bring up CUS223 AP/STA.

3.5 Iperf Parameters

Client is the sender


iperf client

iperf server

Server is the receiver

For iperf server:

```
TCP: iperf -s -w 3M -i 1
UDP: iperf -s -u  -i 1
```

- -i *n* report status every n seconds
- -s start as an iperf server
- -u UDP traffic
- -t m run for m seconds

For iperf client:

```
TCP: iperf -c serverIpAddr -w 3M –i 1 -t 30 -P 9
UDP: iperf -c serverIpAddr -u –b 100m  -i 1 -t 30 -P 9
```

- –w option specifies the TCP window size. The TCP window size is the amount of data that can be buffered during a connection without a validation from the receiver. It can be between 2 and 65,535 bytes.
  – The OS may need to be tweaked to allow buffers of sufficient size. On Linux systems, when specifying a TCP buffer size with the -w argument, the kernel allocates double as much as indicated.
- Parallel transfers may help as well, the –P option can be used for this
- The –b option is used to specify the traffic you want to generate for each thread.

4. Driver Debugging

4.1 Target Data Path Statistics

Physical Device Target Data Stats: the number of times various expected and unexpected transmit and receive events have happened.

Basic command: **iwpriv wlan0 txrx_fw_stats 1**
alternative: **iwpriv wlan0 txrx_fw_mstats 0x1**

Rx Reorder Stats: the number of expected and unexpected events that occurred during receive reordering

Basic command: **iwpriv wlan0 txrx_fw_stats 2**
alternative: **iwpriv wlan0 txrx_fw_mstats 0x2**

Rx Rate Stats: how many times rx frames were received using different rates.

Basic command: **iwpriv wlan0 txrx_fw_stats 3**
alternative: **iwpriv wlan0 txrx_fw_mstats 0x4**

Tx PPDU Log: information about the last several PPDU transmissions.

Basic command: **iwpriv wlan0 txrx_fw_stats 4**
alternative: **iwpriv wlan0 txrx_fw_mstats 0x8**

- “**iwpriv txrx_fw_mstats**” command can be used for uploading multiple stats types together.
- To upload multiple stats types with txrx_fw_mstats, specify an argument that is the bit-OR of the mstats argument for each desired stats type.
- For an example: **iwpriv wlan0 txrx_fw_mstats 0xf**
- Shows all of the four types of statistics.

iwpriv wlan0 txrx_fw_stats 1

```
[  776.389527] ### Tx ###
[  776.389795] comp_queued:  908  #of remote MSDUs completed and put into completion queued
[  776.389957] comp_delivered:  908  # of remote MSDUs in completion queue been sent to host
[  776.390147] msdu_enqued :  955 # of MSDUs queued to WAL. This includes remote/local MSDUs
[  776.390310] wmm_drop :  0     # of MSDUs dropped due to WMM limitation
[  776.390470] local_enqued:  47  # of local MSDUs (non data frames) queued to WAL
[  776.390649] local_freed:  47   # of local MSDUs completed
[  776.390810] hw_queued:  909  # of PPDUs queued to HW
[  776.390969] hw_reaped :  909  # of PPDUs completed from HW
[  776.391157] underrun :  0  # of times Tx underrun happened
[  776.391321] tx_abort :  0
[  776.391480] mpdus_requed :  0  # of MPDUs retried
[  776.391696] excess retries:  0   # of times excess tries happened
[  776.391872] last rc:  231   the last rate code

[  776.392108] ### Rx ###
[  776.392346] ppdu_route_change :  0    # of times for a received PPDU with mixed data /non-data MPDUs.
[  776.392505] status_rcvd :  1990    # of Rx status is used. One Rx status usually represents one MSDU
[  776.392693] r0_frags :  0    # of buffer fragmentation (cross more than one Rx buffer) happened in Ring 0.
[  776.392853] r1_frags:  0   # of buffer fragmentation happened in Ring 1
[  776.393712] r2_frags:  0   # of buffer fragmentation happened in Ring 2
[  776.393925] r3_frags:  0   # of buffer fragmentation happened in Ring 3
[  776.394105] htt_msdus :  1990  # of data MSDUs received
[  776.394313] htt_mpdus:  1990   # of data MPDUs received
[  776.394494] loc_msdus:  1081   # of non-data MSDUs received
[  776.394670] loc_mpdus:  1081   # of non-data MPDUs received
[  776.394880] oversize_amsdu:  0    # of the times that receiving an A-MSDU which has SDUs more than the
size of Rx status ring
```

iwpriv wlan0 txrx_fw_stats 2

Output (STA):

```
[  781.202440] Rx reorder statistics:
[  781.202924]   5 non-QoS frames received  # of MPDUs that came from a peer w/o aggregation configured
[  781.203148]   904 frames received in-order  # of MPDUs received and are in-order, i.e. deliver to upper stack
[  781.203314]   0 frames flushed due to timeout # of MPDUs been flushed (discarded) due to timeout.
[  781.203475]   0 frames flushed due to moving out of window. # of MPDUs been flushed (delivered) due to receiving a new MPDU that moves the reoder window forward.
[  781.203667]   0 frames flushed due to receiving DELBA  # of MPDUs been flushed (discarded) due to DELBA
[  781.203835]   0 frames discarded due to FCS error  # of MPDUs discarded due to FCS error
[  781.203996]   120 frames discarded due to invalid peer  # of MPDUs discarded because we cannot find the corresponding peer.
[  781.204416]   0 frames discarded due to duplication (non aggregation) # of MPDUs came from a peer w/o aggregation configured which are duplication of previous received MPDU
[  781.204583]   0 frames discarded due to duplication in reorder queue # of MPDUs which are duplication of frames in Rx reorder queue
[  781.204583]   0 frames discarded due to processed before  # of MPDUs which are received before. (If the incoming sequence number of a MPDU has more than 2047 offset of expected sequence number in sequence number space, it is considered as processed before.)
[  781.204614]   0 times reorder timeout happened  # of times reorder timer has expired.
```

iwpriv wlan0 txrx_fw_stats 3

Output (STA):

STA side: [88897.679142] WAL RX Rate Info:

[88897.679151] MCS counts (0..9): 0, 0, 0, 0, 34464, 133966, 179145, 274555, 633082, 711998 these are counters for each MCSs 0..9 in case of VHT, and MCS0..7 in the case of the HT association. For 802.11n MCS8..23 please combine this field with NSS field, e.g, MCS8 is NSS 2 MCS0. Please note that this does not capture legacy OFDM/CCK rates.

[88897.679161] SGI counts (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 counters for each SGI enabled MCS.

[88897.679169] NSS counts: 1x1 0, 2x2 8699, 3x3 1958511, 4x4 0 indicate whether 1x1, 2x2 or 3x3 rate is being used. Combined with MCS gives actual (802.11n) MCS in case of HT.

[88897.679175] BW counts: 20MHz 0, 40MHz 1967210, 80MHz 0 indicate number of received frames on 20, 40 and 80MHz. Useful to debug which all BWs are being used currently by the transmitter STA.

[88897.679181] Preamble counts: 221313, 0, 0, 1967210, 0, 0 index 0 counts legacy (CCK/OFDM) ppdus, 1 HT, 2 HT with BF (on peregrine always 0), 3 VHT, 4 VHT with BF (on peregrine always zero), 5 all other, e.g., phy error.

[88897.679188] STBC rate counts (0..9): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 Similar to MCS counts give what all MCSs have STBC enabled.

[88897.679195] LDPC TXBF Counts: 0, 0 the first counter increments for each received LDPC ppdu. Second one increment for each received TxBF frames, which is not supported by Peregrine.

iwpriv wlan0 txrx_fw_stats 4

Output (STA):

```
[  791.199503] Tx PPDU log elements:
[  791.199860]  - PPDU tx to peer 35, TID 0  (intended recipient, traffic type)
[  791.200133]     start seq num = 872, start PN LSBs = 0x368 starting sequence number and starting packet number
[  791.200133]     sent: 1111111111111111111111111111111100000000000000000000000000000000 bitmap of
MPDUs sent
[  791.200133]     ackd: yyyyyyyyyyyyNyyyyyyyyyyNNNyyyyyy-------------------------------- acked MPDUs
[  791.200304]     PPDU is 32 MPDUs, (unknown) MSDUs, 49280 bytes The size of the PPDU in MPDUs, MSDUs,
and bytes
[  791.200491]     enqueued at 51901000, completed at 51910000 (microsec) The time at which the firmware sent the
PPDU to the HW for transmission, and the time at which the HW completed the transmission (received an ack or
reached the specified limit of tries)
[  791.200659]     2 total tries, last tx used rate 231 on 20 MHz chan (flags = 0x0) The number of attempts that were
made until and ack was received, and the rate and channel bandwidth that were used on the final transmission

[  791.200823]  - PPDU tx to peer 35, TID 0
[  791.201007]     start seq num = 872, start PN LSBs = 0x368
[  791.201007]     sent: 0000000000000000000000000000000011111111111111111111111111111111
[  791.201007]     ackd: --------------------------------yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
[  791.201171]     PPDU is 32 MPDUs, (unknown) MSDUs, 49280 bytes
[  791.201333]     enqueued at 51913000, completed at 51913000 (microsec)
[  791.201525]     2 total tries, last tx used rate 231 on 20 MHz chan (flags = 0x0)
…..
```

4.2 Host Data Path Statistics

Host Data Tx/Rx Stats

iwpriv wlan0 txrx_dbg 0x2

Output:

[811.726215] txrx stats:

[811.726401] tx: 908 msdus (1376037 B) the total number of frames and bytes transmitted

[811.726567] rx: 2304 ppdus, 909 mpdus, 904 msdus, 1353842 bytes, 1427 errs number of PPDUs, MPDUs, MSDUs and bytes received. the count of frames that are invalid due to errors

The host data path software is statically compiled to keep various levels of statistics about the data traffic seen. By default the host data software keeps only basic stats, but through recompilation either no stats or details stats can be recorded.

To obtain detailed host data path stats, recompile with TXRX_STATS_LEVEL defined as TXRX_STATS_LEVEL_FULL. This static configuration adds error counters to the host data tx stats, and rx→tx forwarding counters to the host data rx stats:

[7131.002978] txrx stats:

[7131.003411]   tx: sent 2882 msdus (4406647 B), **rejected 0 (0 B), dropped 0 (0 B)**

[7131.003587]   **download fail: 0 (0 B), target discard: 0 (0 B), no ack: 0 (0 B)**

[7131.003748]   rx: 2005 ppdus, 1541 mpdus, 1536 msdus, 2307956 bytes, 1677 errs

[7131.003983]   **forwarded 0 msdus, 0 bytes**

4.3 Target Memory/Register Dump/Change

./athdiag commands and options:

```
--get --address=<target word address>
--set --address=<target word address> --[value|param]=<value>
                                      or --or=<OR-ing value>,  or  --and=<AND-ing value>
--read --address=<target address> --length=<bytes> --file=<filename>
--write --address=<target address> --file=<filename> (--[value|param]=<value>)
--quiet
--device=<device name> (if not default)
```

Examples: Print above help message:

```
  athdiag   or    athdiag --help
```

Dump all of DRAM into /tmp/dram.dump:

```
   athdiag --read --address=0x400000 --length=$((320*1024)) --file=/tmp/dram.dump
```

Disable system sleep and WDT (so that xt-gdb can be used):

```
   athdiag --write --address=0x502c --value=1
   athdiag --write --address=0x4030 --value=0
   athdiag --write --address=0x4040 --value=1
```

4.4 Firmware Debug Log

All the debug log messages generated by firmware are delivered to host as a WMI Event. Host parses these events and prints them on kernel debug buffer.

Format of the Firmware debug log will look like this:

[Time stamp] FWLOG: VAPID* LOG_MESSAGE PARAMS* (* optional elements in the log)

There are **four types** of log levels are supported by the debug log and it is common for all the VAPs and modules. There is no module/VAP specific log level control. However logging for Modules/VAPs can be enabled or disabled. Supported log levels:

DBGLOG_VERBOSE
DBGLOG_INFO
DBGLOG_WARN
DBGLOG_ERROR

Default log level is set to DBG_WARN (meaning logs with log level greater than WARN will be delivered to HOST)

Here are the iwpriv commands to control the debug log.

```
iwpriv wlan# dl_mod_on moduleid
iwpriv wlan# dl_mod_off moduleid
iwpriv wlan# dl_vapon vapid
iwpriv wlan# dl_vapoff vapid
iwpriv wlan# dl_loglevel loglevel
```

4.5 Other Tools

11ac Sniffer

11 AC Tools (Do not enable unless you have marketing approval)
11AC Sniffer

11AC Sniffer
11ac sniffer patch file
11ac sniffer application note

Packetlog

Pktlog tool captures the TX, RX descriptors in real time and provides rates statistics, RSSI statistics, aggregate information, data type description, RA/TA/ BSSID addresses, MPDU sequence numbers, BA, TID, MSDU length and timestamp.

Pktlog tool essentially consists of ath_pktlog.ko (driver), pktlogconf (user space utility) and the pktlogdecoder_11ac.pl perl script.

Thank you!

Qualcomm Atheros Confidential and Proprietary

All data and information contained in or disclosed by this document is confidential and proprietary information of Qualcomm Incorporated and all rights therein are expressly reserved. By accepting this material the recipient agrees that this material and the information contained therein is to be held in confidence and in trust and will not be used, copied, reproduced in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Incorporated.

Qualcomm is a registered trademark and registered service mark of Qualcomm Incorporated. CDMA2000 is a registered certification mark of the Telecommunications Industry Association, used under license. ARM is a registered trademark of ARM Limited. Other products and brand names may be trademarks or registered trademarks of their respective owners. Export of this technology may be controlled by the United States Government. Diversion contrary to U.S. law prohibited.

Product descriptions contained herein are subject to change from time to time without notice.