Atheros CSI Tool User Guide

Yaxiong Xie
xieyaxiongfly@gmail.com
Mo Li
limo@ntu.edu.sg

Wirless And Networked Distributed Sensing (WANDS) system group @ NTU, Singapore

1 TABLE OF CONTENTS

2 SYSTEM REQUIREMENTS

2.1 LINUX SYSTEM

Currently, we implement our Atheros-CSI-Tool on Ubuntu systems. This tool has been tested and works well on Ubuntu 12.04 LTS (64bit) and Ubuntu 14.04 LTS (64bit). We haven't tested it on the latest 15.10, but we believe it should also work well on it. If you have tested it, please kindly let us know your results.

With some engineering effort, this tool can also be implemented on OpenWRT Systems which run on top of WiFi routers (we are currently working on that, and you are welcome to contribute if you have interest).

2.2 LINUX KERNEL

Atheros-CSI-Tool is built into the Linux Kernel 4.1.10. We strongly recommend you to use the same version of Linux kernel as we do. If you are using a different version of Linux kernel you may follow this guide to replace it with Linux Kernel 4.1.10.

2.3 LINUX DRIVER

Atheros-CSI-Tool is built on top of Ath9k, which is an open source Linux kernel driver supporting Atheros 802.11n PCI/PCI-E chips. Theoretically this tool is able to support all Atheros 802.11n WiFi chipsets. Our test experience, however, is based on Atheros AR9580 NIC. If you have tested it on other types of Atheros NICs, please kindly inform us whether it is successful or let us know the problems that you encounter.

3 INSTALL THE LINUX KERNEL

3.1 INSTALLATION OF THE REQUIRED PACKAGES

We need to install some necessary packages before downloading and installing our modified kernel.

To install git to download the latest version of Atheros-CSI-Tool.

```
sudo apt-get install git
```

Some packages are need for successful running of "make menuconfig"

```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

Finally, install some packages for compiling the Hostapd.

```
sudo apt-get install libnl-dev libssl-dev
```

3.2 KERNEL DOWNLOADING AND INSTALLATION

```
git clone https://github.com/xieyaxiongfly/Atheros-CSI-Tool.git
```

When it finished, you should find a folder named "Atheros-CSI-Tool L". Enter it and we will be in the folder "~/Atheros/Atheros-CSI-Tool/". Next step is to configure the kernel and generate the ".config" file. Run the following command in this folder:

```
make menuconfig
```

We will use the default config provided by the kernel.

Next, compile the kernel modules.

```
make -j16
make modules
sudo make modules_install
sudo make install
```

Note here that the "-j16" is used for accelerating the compiling process. The number "16" is select according to the number of your CPU-core. Generally, 2 times of your CPU-core.

Finally, reboot the system.

```
sudo reboot
```

We are done! You have implemented our customized Linux Kernel 4.1.10. Run the following command to check your kernel version.

```
uname -r
```

If the output is "4.1.10+", then your installation is successful.

4 RECEIVING CSI

You need to enable the CSI receiving functionality at receiver side and then transmit HT packets from another device to the receiver. The receiver will upload the CSI of the HT packets to MAC layer and our program will record it automatically.

4.1 DATA STRUCTURE

The followings are several important data structure used in our program.

➢ **Receiving buffer**

```
unsigned char buf_addr[4096];
```

This is the buffer we use to communicate with kernel. If the kernel receives a packet and calculates its CSI, it will put the CSI, payload and packet status information to this buffer.

➢ **Payload buffer**

```
unsigned char data_buf[1500];
```

This is the buffer we use to store the payload of the received packet. We set its size to 1500 bytes, but you can change it according to your packet size.

➢ **Complex number**

```
typedef struct
{
    int real;
    int imag;
}COMPLEX;
```

This is the data structure we use to store the complex number of channel state.

➢ **CSI matrix**

```
csi_matrix = COMPLEX[nr_max][nc_max][114];
```

We use a COMPLEX number array to store the CSI matrix we received, where "nr_max"/"nc_max" means the maximum number of rx/tx antenna and 114 is the subcarrier number of a 40MHz channel. We set "nr_max=3" and "nc_max = 3".

This matrix is able to store the CSI of one packet transmitted using at most 3 tx and 3 rx antennas and 40MHz channel bandiwidth. If you use less than 3 antennas, this matrix will not be fully occupied.

➢ **Packet status structure**

```
 typedef struct
{
    u_int64_t    tstamp;         /* h/w assigned time stamp */

    u_int16_t    channel;        /* wireless channel (represented in Hz)*/
    u_int8_t     chanBW;         /* channel bandwidth (0->20MHz,1->40MHz)*/

    u_int8_t     rate;           /* transmission rate*/
    u_int8_t     nr;             /* number of receiving antenna*/
    u_int8_t     nc;             /* number of transmitting antenna*/
    u_int8_t     num_tones;      /* number of tones (subcarriers) */
    u_int8_t    noise;        /* noise floor (to be updated)*/

    u_int8_t    phyerr;       /* phy error code (set to 0 if correct)*/

    u_int8_t    rssi;         /*  rx frame RSSI */
    u_int8_t    rssi_0;       /*  rx frame RSSI [ctl, chain 0] */
    u_int8_t    rssi_1;       /*  rx frame RSSI [ctl, chain 1] */
    u_int8_t    rssi_2;       /*  rx frame RSSI [ctl, chain 2] */

    u_int16_t   payload_len;  /*  payload length (bytes) */
    u_int16_t   csi_len;      /*  csi data length (bytes) */
    u_int16_t   buf_len;      /*  data length in buffer */
}csi_struct;
```

This structure is used to store the packet status information.

- **tstamp**: it stores the time (TSF value) when packet is received, expressed in **μs**
- **channel**: it stores center frequency of the wireless channel, expressed in **MHz**.
- **chanBW**: it stores the channel bandwidth. It is 20MHz if set to 0 and 40MHz if set to 1.
- **rate**: it stores the data rate of the received packet. Its value is a unsigned 8 bit integer number and the mapping between this value and the rate choice of 802.11 protocol, please refer to Appendix Ⅰ.
- **nr**: it stores the number of receiving antenna.
- **nc**: it stores the number of transmitting antenna.
- **num_tones:** it stores the number of subcarrier that used for data transmission.
- **noise:** it stores the noise floor, expressed in dB. But it needs to be update and is set to 0 in current version.
- **phyerr:** it stores the phy error code, set to 0 if correctly received. For detailed error type, please refer Appendix Ⅱ.
- **rssi:** it stores the rssi of combination of all active chains.
- **rssi_0, rssi_1, rssi_2:** it stores the rssi of active chain 0, chain 1 and chain 2.
- **payload_len**: it stores the payload length of received packet, expressed in bytes.
- **csi_len**: it stores the csi data length in the received data buffer, expressed in bytes.
- **buf_len**: it stores the total data length in the received data buffer, expressed in bytes.

4.2 APIs

We introduce the API to receive CSI from kernel.

➢ **Reading the receiving buffer**

```
int read_csi_buf(unsigned char* buf_addr,int fd, int BUFSIZE)
```

This function listens and tries to read the receiving buffer. If the kernel receives a packet and calculates the CSI, it will upload the CSI, payload and packet information to the receiving buffer. This function will read the buffer immediately after the CSI is uploaded. A timer is set and when it times out, this function will read the buffer and check whether the buffer is empty or not.

After reading the receiving buffer, this function will return a number which indicates how many bytes are there in the buffer. If the number is 0, it means no CSI is uploaded.

➢ **Recording the packet status**

```
void record_status(unsigned char* buf_addr, int cnt, csi_struct* csi_status)
```

This function fetches the data in the receiving buffer and fills the packet status to the packet status structure. From the packet status structure, you have access to all the information of received packet.

This function is normally called after the function "`read_csi_buf`" returns a non-zero number, which means that the receiving buffer is not empty and the CSI is uploaded.

- **Recording the CSI and payload**

```
void record_csi_payload(unsigned char* buf_addr,
            csi_struct* csi_status, unsigned char* data_buf,
            COMPLEX(* csi_matrix)[3][114])
```

This function fetches the data in the receiving buffer and fills the CSI to the csi matrix and fills the payload of the received packet to the payload buffer.

This function is normally called after the function "read_csi_buf" returns a non-zero number, which means that the receiving buffer is not empty and the CSI is uploaded.

- **Processing csi and payload**

```
void  porcess_csi(unsigned char* data_buf, csi_struct* csi_status,
            COMPLEX(* csi_buf)[3][114])
```

This function takes the csi matrix, payload buffer, packet status structure as input. The CSI, payload can be processed by this function. In current implementation, we leave this function blank.

4.3 Example

We build an example for receiving CSI, i.e., "/Atheros-CSI-Tool-UserSpace-APP/recvCSI/main.c".

4.4 Off-line processing

We also support off-line CSI processing by logging the CSI information in a log file.

Every time the kernel uploads CSI to the receiving buffer, the buffered data are recorded into a file. We build an example in the file "/Atheros-CSI-Tool-UserSpace-APP/recvCSI/main.c" for your reference.

An example for the logged CSI .

```
csi = timestamp: 865834005
        csi_len: 420
        channel: 2437
       err_info: 0
    noise_floor: 0
           Rate: 133
      bandWidth: 0
      num_tones: 56
             nr: 3
             nc: 1
           rssi: 46
          rssi1: 35
          rssi2: 38
          rssi3: 45
    payload_len: 1040
            csi: [3x1x56 complex double]
        payload: [1040x1 uint8]
```

We store the CSI matrix, payload and packet status information in the same structure. From this structure, we transmit this packet using 3 rx antennas, 1 tx antenna and 20MHz chanel. Hence, the csi matrix is 3x1x56. The payload length is 1040 byte. The channel is 2437MHz (2.437GHz). The data rate is 133, and according to Appendix I, we could find that we are using 52Mbps.

5 AP CONFIGURATION

Atheros-CSI-Tool requires two computers which have our customized Linux Kernel 4.1.10 installed. You need to make one of the computers in to the 802.11n AP mode. We can configure a PC into an 802.11 AP using Hostapd.

5.1 INSTALLATION OF THE HOSTAPD

You may fetch the source code of our UserSpace application source from our GitHub. Enter the folder you want to put the UserSpace source code e.g., in "~/Atheros/". Run the following code:

```
git clone https://github.com/xieyaxiongfly/Atheros-CSI-Tool-UserSpace-APP.git
```

When it finished, you should find a folder named "Atheros-CSI-Tool -UserSpace-APP". Enter Hostapd file folder: "~/Atheros/Atheros-CSI-Tool-UserSpace-APP/hostapd-2.5/hostapd/". Compiling the source code:

```
make
```

If the compiling process succeeds, the Hostapd software is ready. There are still a few more steps to make it work.

5.2 CONFIGURATION OF THE WIFI INTERFACE

Use "ifconfig" to find the interface name of your WiFi card, e.g., "wlan0".

Edit the file "/etc/network/interfaces":

```
auto wlan0
iface wlan0 inet static
address 10.10.0.1
netmask 255.255.255.0
```

5.3 INSTALLATION OF THE ISC-DHCP-SERVER

You may install a DHCP server to automatically assign IP addresses for the WiFi clients, e.g., ISC DHCP server, and it can be installed using the following command:

```
sudo apt-get install isc-dhcp-server
```

Change the default interface of ISC-DHCP-Server to the WiFi card. Edit the file "/etc/default/isc-dhcp-serve" and set INTERFACES:

```
INTERFACES="wlan0"
```

Edit the DHCP server configuration file "/etc/dhcp/dhcpd.conf".

```
#option definitions common to all supported networks…
#option domain-name “example.org”;
#option domain-name-servers ns1.example.org, ns2.example.org;
#default-lease-time 600;
#max-lease-time 7200;
```

At the end of the file you may add:

```
subnet 10.10.0.0 netmask 255.255.255.0 {
        range 10.10.0.2 10.10.0.16;
        option domain-name-servers 8.8.4.4, 208.67.222.222;
        option routers 10.10.0.1;
}
```

There is a batch script file named "start_hostapd.sh" in "~/Atheros/Atheros-CSI-Tool-UserSpace-APP/hostapd-2.5/hostapd/". Run it to start the Hostapd.

6 APPENDIX I

To map the rate reported by Atheros-CSI-Tool to a real 802.11n rate, please refer to the table below. NOTICE: this table only contains the HT rate.

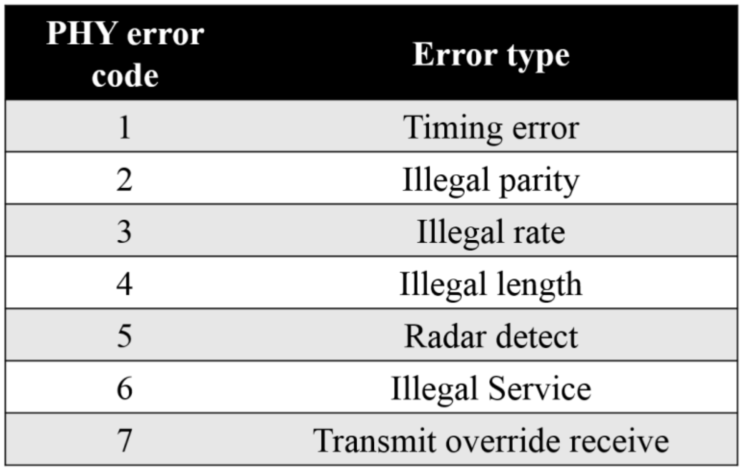
Rate(Hex)	Rate(Decimal)	Stream	HT20 (800ns GI)	HT40 (800ns GI)
0x80	128	1	6.5	13.5
0x81	129	1	13	27
0x82	130	1	19.5	40.5
0x83	131	1	26	54
0x84	132	1	39	81
0x85	133	1	52	108
0x86	134	1	58.5	121.5
0x87	135	1	65	135
0x88	136	2	13	27
0x89	137	2	26	54
0x8A	138	2	39	81
0x8B	139	2	52	108
0x8C	140	2	78	162
0x8D	141	2	104	216
0x8E	142	2	117	243
0x8F	143	2	130	270
0x90	144	3	19.5	40.5
0x91	145	3	39	81
0x92	146	3	58.5	121.5
0x93	147	3	78	162
0x94	148	3	117	243
0x95	149	3	156	324
0x96	150	3	175.5	364.5
0x97	151	3	195	405

For Legacy rate, please refer to the following table.

Rate(Hex)	Rate(Decimal)	Legacy Rate	Protocol
0x8	8	48Mbps	OFDM
0x9	9	24Mbps	OFDM
0xA	10	12Mbps	OFDM
0xB	11	6Mbps	OFDM
0xC	12	54Mbps	OFDM
0xD	13	36Mbps	OFDM
0xE	14	18Mbps	OFDM
0xF	15	9Mbps	OFDM
0x18	24	11Mbps_L	CCK
0x19	25	5.5Mbps_L	CCK
0x1A	26	2Mbps_L	CCK
0x1B	27	1Mbps_L	CCK
0x1C	28	11Mbps_S	CCK
0x1D	29	5.5Mbps_S	CCK
0x1E	30	2Mbps_S	CCK

7 APPENDIX II

To understand the real type of phy error, please refer to the following table.



PHY error code	Error type
1	Timing error
2	Illegal parity
3	Illegal rate
4	Illegal length
5	Radar detect
6	Illegal Service
7	Transmit override receive

-