ESP8266 Sniffer Introduction

Version 0.3

Espressif Systems IOT Team

Table of Contents

1. Sniffer Introduction

ESP8266 can enter promiscuous mode (sniffer) and capture IEEE 802.11 packets in the air.

The following HT20 packets are support:

- 802.11b
- 802.11g
- 802.11n (from MCS0 to MCS7)
- AMPDU types of packets

The following are not supported:

- HT40
- LDPC

Although ESP8266 can not completely decipher these kinds of IEEE80211 packets completely, it can still obtain the length of these special packets.

In summary, while in sniffer mode, ESP8266 can either capture completely the packets or obtain the length of the packet:

- Packets that ESP8266 can decipher completely; ESP8266 returns with the
  - MAC address of the both side of communication and encryption type and
  - the length of entire packet.
- Packets that ESP8266 can only partial decipher; ESP8266 returns with
  - the length of packet.

Structure `RxControl` and `sniffer_buf` are used to represent these two kinds of packets. Structure `sniffer_buf` contains structure `RxControl`.

```
struct RxControl {
    signed rssi:8;              // signal intensity of packet
    unsigned rate:4;
    unsigned is_group:1;
    unsigned:1;
    unsigned sig_mode:2;        // 0:is 11n packet; 1:is not 11n packet;
    unsigned legacy_length:12; // if not 11n packet, shows length of packet.
    unsigned damatch0:1;
    unsigned damatch1:1;
    unsigned bssidmatch0:1;
```

```
        unsigned bssidmatch1:1;
        unsigned MCS:7;            // if is 11n packet, shows the modulation
                                   // and code used (range from 0 to 76)
        unsigned CWB:1; // if is 11n packet, shows if is HT40 packet or not
        unsigned HT_length:16;// if is 11n packet, shows length of packet.
        unsigned Smoothing:1;
        unsigned Not_Sounding:1;
        unsigned:1;
        unsigned Aggregation:1;
        unsigned STBC:2;
        unsigned FEC_CODING:1; // if is 11n packet, shows if is LDPC packet or not.
        unsigned SGI:1;
        unsigned rxend_state:8;
        unsigned ampdu_cnt:8;
        unsigned channel:4; //which channel this packet in.
        unsigned:12;
    };

    struct LenSeq{
        u16 len; // length of packet
        u16 seq; // serial number of packet, the high 12bits are serial number,
                 //    low 14 bits are Fragment number (usually be 0)
        u8 addr3[6]; // the third address in packet
    };

    struct sniffer_buf{
        struct RxControl rx_ctrl;
        u8 buf[36 ]; // head of ieee80211 packet
        u16 cnt;     // number count of packet
        struct LenSeq lenseq[1];  //length of packet
    };

    struct sniffer_buf2{
        struct RxControl rx_ctrl;
        u8 buf[112];
        u16 cnt;
        u16 len;  //length of packet
    };
```

Callback `wifi_promiscuous_rx` has two parameters (`buf` and `len`). `len` means the length of `buf`, it can be: `len` = 128, `len` = X * 10, `len` = 12 :

Case of LEN == 128

- `buf` contains structure `sniffer_buf2`: it is the management packet, it has 112 bytes data.
- `sniffer_buf2.cnt` is 1.
- `sniffer_buf2.len` is the length of packet.

Case of LEN == X ∗ 10

- `buf` contains structure `sniffer_buf`: this structure is reliable, data packets represented by it has been verified by CRC.
- `sniffer_buf.cnt` means the count of packets in `buf`. The value of `len` depends on `sniffer_buf.cnt`.
  - `sniffer_buf.cnt==0`, invalid buf; otherwise, `len = 50 + cnt ∗ 10`
- `sniffer_buf.buf` contains the first 36 bytes of ieee80211 packet. Starting from `sniffer_buf.lenseq[0]`, each structure `lenseq` represent a length information of packet. `lenseq[0]` represents the length of first packet. If there are two packets where `(sniffer_buf.cnt == 2)`, `lenseq[1]` represents the length of second packet.
- If `sniffer_buf.cnt > 1`, it is a AMPDU packet, head of each MPDU packets are similar, so we only provide the length of each packet (from head of MAC packet to FCS)
- This structure contains: length of packet, MAC address of both sides of communication, length of the head of packet.

Case of LEN == 12

- `buf` contains structure `RxControl`; but this structure is not reliable, we can not get neither MAC address of both sides of communication nor length of the head of packet.
- For AMPDU packet, we can not get the count of packets or the length of packet.
- This structure contains: length of packet, `rssi` and `FEC_CODING`.
- `RSSI` and `FEC_CODING` are used to guess if the packets are sent from same device.

Summary

We should not take too long to process the packets. Otherwise, other packets may be lost.

The diagram below shows the format of a ieee80211 packet:

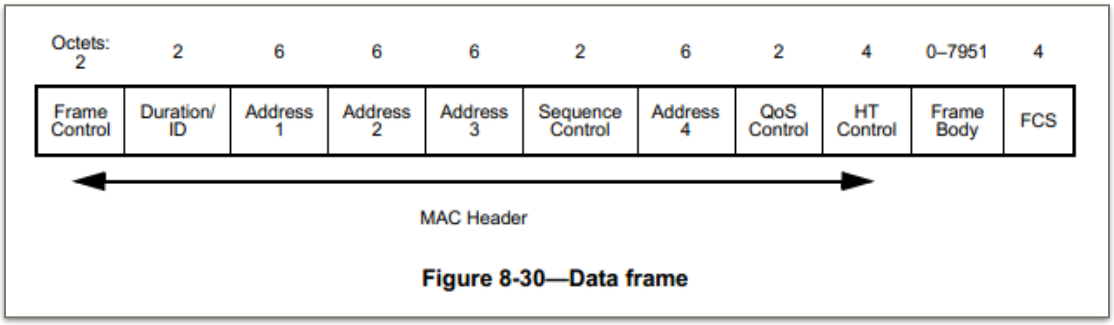

Figure 8-30—Data frame

- The first 24 bytes of MAC Header of data packet are needed:
  - `Address 4` field depends on `FromDS` and `ToDS` which is in `Frame Control`;
  - `QoS Control` field depends on `Subtype` which is in `Frame Control`;
  - `HT Control` field depends on `Order Field` which is in `Frame Control`;
  - More details are found in IEEE Std 80211-2012.
- For WEP packets, MAC Header is followed by 4 bytes IV and before FCS there are 4 bytes ICV.
- For TKIP packet, MAC Header is followed by 4 bytes IV and 4 bytes EIV, and before FCS there are 8 bytes MIC and 4 bytes ICV.
- For CCMP packet, MAC Header is followed by 8 bytes CCMP header, and before FCS there are 8 bytes MIC.

2. Sniffer Application Scenarios

Because some APs won't transmit UDP broadcast packets to WLAN, so only the UDP packets from mobile phone can be listened. These UDP packets are from mobile phone to AP, and are encrypted.

Scenario 1: IOT_device can get all packets from mobile phone

This scenario requires:

- The connection between mobile phone and AP is working in 802.11b, or 802.11g, or 802.11n HT20 mode
- The distance between mobile phone and AP is longer than the distance between mobile phone and IOT_device

IOT-device firmware can set filter of MAC address or MAC-header (include MAC-cryption-header), it can also set a filter for retransmission.

Meanwhile, for 802.11n AMPDU packets, IOT_device can also get the length of packet and MAC-header (include MAC-cryption-header)

Scenario 2: IOT_device can not get all packets from mobile phone, signal is strong, but packet format is not supported.

Case 1:

The distance between mobile phone and AP is much longer than the distance between mobile phone and IOT_device. Then the high-frequency packets from mobile phone can be got by AP, but can not be got by IOT_device.

For example, mobile phone sent MCS7 packets which can be got correctly by AP, but IOT_device can only parse its packet header of physical layer (HT-SIG), because packet header of physical layer is encoded on low-speed (6Mbps).

Case 2:

Format of packets that mobile phone sent to AP is not supported by IOT_device, such as:

- HT40;
- LDPC ;
- 11n MCS8 and later version, such as MIMO 2x2。

IOT_device can not get the whole packet, but can parse its packet header of physical layer (HT-SIG).

In both case 1 and case 2, IOT_device can get HT-SIG which include the length of packet in physical layer. Please pay attention on following items when using it:

- When it isn't AMPDU packet or only one sub-frame in AMPDU packet, the length of UDP packet can be speculated. If the time interval of UDP packets which sent from phone APP is long (20ms ~ 50ms) , each UDP packet will in different packets in physical layer, may be a AMPDU packet which only has one sub-frame.
- Firmware of IOT_device can filter packets from other devices according to RSSI.
- Packet of retransmission need to be filter according to the packets sequence, it means that length of packets which sent consecutively need to be different. For example:
  - Two useful packets can be separated by a specific packet. The specific packet works like separative sign.
  - Length of packet in odd number to be 0~511, length of packet in even number to be 512~1023。

2. Phone APP

For Scenario 2, phone APP should notice:

- Time interval of each UDP packet to be longer than 20ms
- Two data packets can be separated by a specific packet. The specific packet works like separative sign.
- Packet with redundant data so that packet can verify each other.
- Set flag-packet at the beginning of sequence. Then phone APP can be cyclic sending the whole sequence.
- Only need to send the lowest 2 bytes of AP's BSSID (MAC address), IOT-device can still get it. If AP will broadcast its SSID, then phone APP need not to send AP's SSID either. So AP beacon need to be analyzed to check if the AP will broadcast its SSID.
- Length of UDP packet need to be multiply by 4. Because when phone APP sent a AMPDU packet which only has one sub-frame, packet length will be filled to be a multiple of 4.

For Scenario 1, phone APP can send packets as fast as possible.

Phone APP won't know it is Scenario 1 or Scenario 2 for IOT_device.

3. IOT-device Firmware

For Scenario 2, IOT-device should notice:

- Search the channel which has strongest signal first, according to RSSI.
- Filter useless packets according to RSSI. Considering 10~15db fluctuations in the air, some packets may be decline 10db or more. We could search the strongest signal at first, then extend the range since find the target sequence.
- Check the Aggregation bit of HT-SIG to distinguish AMPDU packet.
- AMPDU packet can only be encrypt by CCMP(AES).
- To design the length of packet that works as separative sign, different QoS, different encryption algorithm and AMPDU packet will be a multiple of 4, all of these should be taken into consideration.
- Use relative value to transmit information, for example, the value that the length of data packet minus the length of packet that works as separative sign.