Passive Scan on network processor CC3100

Ilya Sukhoplyuev
Innopolis SNE Student
Email: i.sukhoplyuev@innopolis.university

Luis Funes
Innopolis SNE Student
Email: j.funes@innopolis.university

***Abstract*—The CC3100 driver does not support the passive scan functionality. For this reason, the L1/L2 transceiver mode of CC3100 can be used to obtain the basic information from the advertisements of access points working nearby. This research describes the passive scan implementation and the power consumption comparing it with the active scan which is built-in the CC3100 network processor.**

I. Introduction

Before establishing a Wi-Fi connection, the client device should discover the nearest access points. This process is called *scan* and the IEEE 802.11 standard provides several options to implement this.

In the first option, the end device can send a broadcast message. After the access point receives this message, it should send an advertisement as a response. On the second option, the device can listen to the medium for access point advertisements. These ways are called *active and passive scan* respectively.

While active scan provides interactive behavior, passive scan brings a confidentiality advantage: with it, a user has the capability of hiding its Wi-Fi device as long as there is no on-going communication. For example, a sensor, such as a voice recorder or video camera, collects the user data to its internal memory and a user can eventually pull that data using Wi-Fi connection. At the same time, it is important that the device does not emit any signals to avoid being detected from others.

Active scan is not appropriate in this kind of cases, because the user cannot control when the device sends broadcast messages. In the other side, the user can handle it with the passive scan by creating an ad-hoc network to interact with the sensor. That network would notify the sensor to start the connection. Meanwhile, when the network is not running, the sensor passively checks the network status.

II. Problem definition

Although the passive scan might be required, it is not necessarily available in some devices. For example, the driver of CC3100[10] — network processor developed by Texas Instruments (TI) — does not support this function. On the other hand, the driver allows us to use CC3100 in the transceiver mode. By using this, we can parse frames at a low-level from the Main Control Unit (MCU) side and implement the passive scan by ourselves.

III. Research questions and goals

- What is the advertisement structure? How can we extract the network's name from this advertisement?
- How does the CC3100 work in transceiver mode?
- Can we use the CC3100 as a sniffer?
- How does the active scan functionality of CC3100 work?
- Implement the passive scan functionality by listening to the wireless medium and analyzing the received packets.
- Compare the power consumption between active and passive scans.

IV. Background

A. Used hardware

The CC3100 network processor is produced by TI to bring Wi-Fi and Internet stack functionalities to Internet-of-Things (IoT) devices within one chip. This network processor connects to the Main Control Unit (Host MCU) with Serial Peripheral Interface (SPI) or Universal Asynchronous Receiver-Transmitter (UART).

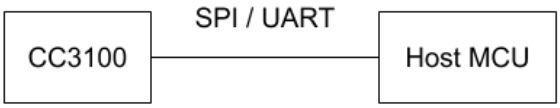

Figure 1. The concept of using CC3100

For evaluation purposes, the TI provides various electronic kits. We used the CC3100 BoosterPack[11] and Advanced Emulation Kit for CC31XX BoosterPack[6]. Together they allow a developer to connect the CC3100 with a PC via USB.

To use them, TI provides the CC3100 Software Development Kit (SDK), which contains SimpleLink driver, SimpleLink Studio and C-language examples. The SimpleLink driver is a hardware-independent code which allows a programmer to control CC3100, whereas the SimpleLink Studio is the SimpleLink driver port to the Windows OS. Using this software together, the programmer can develop C-applications on the PC as on Host MCU side (Figure 2).

B. Wi-Fi description

The Wi-Fi or standard IEEE 802.11[3] describes the set of physical layers and data-link layer in terms of the *Open Systems Interconnection model* (OSI). Each type of physical layer defines the using ISM band (*Industrial, Scientific and Medical Radio band*) and *electromagnetic modulations*. The

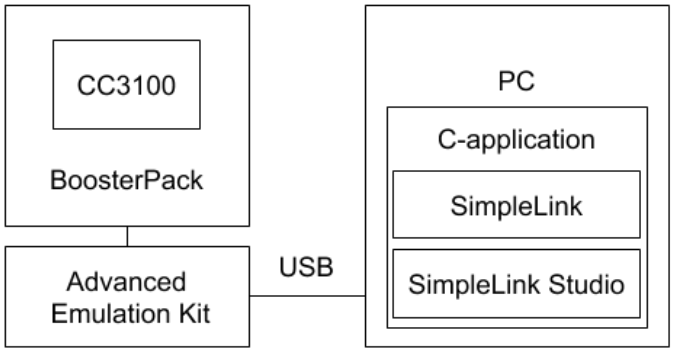

Figure 2. The used hardware configuration

modulation describes the coding and decoding methods of digital frames into electromagnetic waves. Depending on these methods, each modulation can have one or more *data rates* which are set the maximum available speeds of data transmission. Inside the ISM band, the standard determines the set of *channels* in which transmission happens.

For example, CC3100 supports the 802.11 b/g/n standards which means the transceiver operates at 2.4GHz band, where it can work on one of the 13 available channels. The **n**-standard — as well as **b** and **g** standards — requires the implementation of the *Direct Sequence Spread Spectrum (DSSS)* and *Orthogonal Frequency Division Multiplexing (OFDM)* modulations, which allows the microchip to reach data rates from 1 to 54Mbps.

On the data-link layer 802.11 standard describes *Media Access Control (MAC)* and *Link-Local Control (LLC)* sub-layers to make communication through wireless medium similar to the local wired networks for the digital devices. We consider the most popular scenario where Wi-Fi network is organized around one or more *Access Points (AP)* and *Client Devices (or Stations, STA)*. Each network has a human-readable name called *Service Set Identifier (SSID)*. By knowing the SSID and security parameters, a client station can associate and authenticate with an AP and start to transmit user-data through the wireless network. This process between a station and an AP can be considered an analog of a wired connection between a PC and a switch.

To check the network availability the client device sends the *probe request* during active scan procedure. The request contains the SSID of the requested network and the wireless capabilities of the client. The SSID field may be empty (having a zero length) so-called *broadcast SSID*. Using this, the station is requesting all available networks in the area. When the AP receives that request, it replies with the *probe response*, which contains all the appropriate parameters for this AP and the station to negotiate the connection. These parameters include the rates, channel(s), SSID, security configuration and others.

Another way to find surrounding networks is when a client device awaits for AP broadcast advertisements. These are called *beacons*, which contain the public parameters of this AP. Its content is similar to the probe response with a slight difference: the AP does not know the client's possibilities, so the beacon only describes the capabilities of the AP and the client should decide if it fits or not. The frequency of beacon advertisements can be configured by an AP owner. This interval is measured in Time Unit, which represents 1024 microseconds according to the standard. The APs are often pre-configured with 100 TUs, which means that beacons are sent every 102.4 milliseconds.

V. Beacon Frame Format

In the IEEE 802.11 standard[3] are defined 3 major types of frames:

1) Data Frame;
2) Control Frame;
3) Management Frame.

They share the *General Frame Format*, a sample of this is observed in figure 3. In this structure we can find the following fields: *frame control*, *duration/ID*, *address 1* and *frame check sequence (FCS)*. These are considered the minimal frame format and are present in every frame.

Octets: 2	2	6	0 or 6	0 or 6	0 or 2	0 or 6	0 or 2	0 or 4	variable	4
Frame Control	Duration /ID	Address 1	Address 2	Address 3	Sequence Control	Address 4	QoS Control	HT Control	Frame Body	FCS

Figure 3. General frame structure

We can find that the *frame control* is formed by 2 octets and contains other fields (figure 4). The most interesting fields for us are *Type* and *Sub-type* with 2 and 4 bits respectively. Having that for the *Type* value when this bits are as 00_2 will corresponds to a *Management Frame* type, when the values are 01_2 corresponds to a *Control Frame* and having 10_2 will corresponds to a *Data Frame*, the value 11_2 is reserved.

B0 B1	B2 B3	B4 B7	B8	B9	B10	B11	B12	B13	B14	B15
Protocol Version	Type	Subtype	To DS	From DS	More Frag-ments	Retry	Power Management	More Data	Protected Frame	+HTC/ Order
Bits: 2	2	4	1	1	1	1	1	1	1	1

Figure 4. The frame control structure

For the *Sub-type value* we can have a wider variety of options, some of them as, probe request specified as sub-type value 0100_2, probe Response as sub-type value 0101_2 and the beacon is a management frame with sub-type value of 1000_2. The structure of the management frame is presented in figure 5.

Octets: 2	2	6	6	6	2	0 - 2312	4
Frame Control	Duration	Address 1 (DA)	SA	BSSID	Sequence Control	Frame Body	FCS
		MAC Header					

Figure 5. Management Frame

Inside of the *Beacon Frame* we can find some other fields that will be described briefly, starting from the address fields which are used to identify the `Basic Service Set ID`

(BSSID) and `MAC Addresses` of the devices involved in the communication, typically 3 addresses are used, their contents are:

- Address 1: Destination Address (DA);
- Address 2: Source Address (SA);
- Address 3: BSSID.

The *frame body* of the management frame consists of *fields* and *elements* that appear in the specified order. For the beacon the fields are:

Field name	Octets	Description
Timestamp	8	AP up-time, in mcsec
Beacon Interval	2	Time between beacons, in TUs
Capability Information	2	AP capabilities

Each *element* shares the general binary structure: one octet *Element ID* and second octet *Length* which contains size in octets of the next *Information field*(figure 6). Depending on *Element ID*, an additional field *Element ID extension* might exist, which allows to define more *element* types and carry additional information in the beacon frame. That structure gives the possibility to create a configurable list of data and WI-Fi vendors can extend it if required.

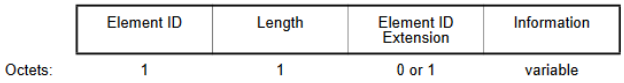

Figure 6. The general element format

In the paper we concentrate only on the first *elements*:

Element name	Element ID	Length, octets	Description
Service Set Identifier (SSID)	0	0-32	Network name
Supported rates	1	1-8	Each octet represents one supported rate

The detailed example of beacon structure can be found in the appendix A.

VI. Implementation

The implementation process is split into separate programs. Each of them meant for testing the transceiver mode according to the programming User Guide[9]. Then we adjust these programs for analyzing the beacon frames and implementing the passive scan.

A. Low-level communication

The first experience with CC3100 was testing how it communicates on layer 2. With a special socket, the driver allows sending a custom frame with the chosen rate and channel. We verified it with the regular computer running Linux with the Wi-Fi board configured in the monitor mode.

After that, we received the frame from the second CC3100 board. In that stage, the main task was to distinguish the sent frames from others. We sent the frame with a pre-configured destination MAC address: `00:23:75:55:55:55`. On the receiving side, the program checked this and discarded the others non-matching frames. During the experiment, we noticed that CC3100 contains a built-in mechanism to detect the rate of received frames. Meanwhile, the channel should be configured as one of the socket's options.

Finally, we used the previous program to make the CC3100 devices exchange the messages in which they count from 1 to 100 together. In the low-level communication there was no acknowledgment mechanism by default, so we had to implement a simple retransmission-and-reply procedure. From one side, after the device sent one packet, it expected the answer during the next 20 frames. If the device did not find the answer frame, then it re-transmits the packet and waits for an answer again. The device from the other side had a similar behavior after receiving the first message.

B. Sniffer example with WireShark

Next program was a sniffer using CC3100 in transceiver mode. This use-case is in the programming User Guide[9], but we had to recover missing C-structures not mentioned in the guide.

This sniffer program redirects the receiving frames to WireShark through the pipe. These frames have to be structured in the pcap-file format to make WireShark understand them. According to the WireShark Wiki[12], the program provides a global pcap header which introduces the data as pcap-file, then put the pcap-entry header which will describe the timestamp and length of the following frame. In addition to IEEE 802.11 LAN frame which the CC3100 gives, we provide additional radiotap-header[4]. Using that, the program can specify physic information relevant to the packet.

```
IEEE 802.11 Beacon frame, Flags: ........
   Type/Subtype: Beacon frame (0x0008)
 > Frame Control Field: 0x8000
   .000 0000 0000 0000 = Duration: 0 microseconds
   Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
   Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
   Transmitter address: AsustekC_ec:6f:18 (bc:ee:7b:ec:6f:18)
   Source address: AsustekC_ec:6f:18 (bc:ee:7b:ec:6f:18)
   BSS Id: AsustekC_ec:6f:18 (bc:ee:7b:ec:6f:18)
   .... .... .... 0000 = Fragment number: 0
   0011 1011 1111 .... = Sequence number: 959
IEEE 802.11 wireless LAN
 v Fixed parameters (12 bytes)
      Timestamp: 0x000001120bdc013b
      Beacon Interval: 0.102400 [Seconds]
   > Capabilities Information: 0x0411
 v Tagged parameters (246 bytes)
   > Tag: SSID parameter set: CC3100SNE
```

Figure 7. Part of captured beacon frame by CC3100

This sniffer helps to analyze and explore the frames in practice. The partial example is in figure 7. A more detailed example with explanation can be found in the appendix A.

C. Active Scan program

The last preliminary program was using default active scan functionality to print the available networks. In the SDK, there is a Scan Policy Application[7], part of which demonstrates the required actions to trigger the active scan. To do it, the Host MCU sets up the time interval during which the active scan should happen. Then host MCU idles 1s and then retrieves the list of up to 20 entities. Each entry represents the available network with its BSSID, SSID, RSSI, and security type (figure 8).

```
      BSSID             RSSI    SSID
00:08:2F:4A:90:23       -67     UniversityStaff
BC:67:1C:E8:FB:63       -79     UniversityStaff
BC:67:1C:E8:FF:41       -75     InnoMeeting
BC:67:1C:E8:FB:67       -79     UniversityGuest
BC:67:1C:E8:FF:47       -76     UniversityGuest
00:08:2F:4A:90:27       -67     UniversityGuest
00:08:2F:4A:90:21       -67     InnoMeeting
00:08:2F:32:F3:D8       -50     UniversityStudent
58:97:1E:A2:60:B8       -80     UniversityStudent
00:08:2F:32:F3:D0       -50     Innopolis
58:97:1E:A2:60:B0       -80     Innopolis
```

Figure 8. The results of active scan program

D. Passive Scan Program

The passive scan application sets the CC3100 in transceiver mode and listens on 13 available channels. When it receives a frame, the program determines the type and subtype. If they match to beacon type, then the program includes the new information about the network nearby.

The parsing of a binary frame is simple with C-language. First, the beacon is described in the C-structures, and then the frame buffer is cast into that structure. After that, every described field becomes accessible by its name. Especially this situation proves with bit-fields structure [2], which allows the manipulation with separate bits of Control Field as shown in listing 1.

The beacon interval is configurable from the AP, so the time of waiting for a beacon frame should be related to this parameter. This way, the passive scan function has the duration time argument, which will set depending on the device configuration.

This time span is distributed into 13 channels. As mentioned, the default beacon interval is 102.4 milliseconds. So to ensure we catch at least one beacon on each channel, we should make a passive scan during $1300 - 1400$ milliseconds.

In our environment, there is a Wi-Fi infrastructure with a set of access points configured to act as one network. And there are several networks inside that infrastructure. So, we grouped each network by SSID for better visualization. The example of one network can be seen in figure 9.

Listing 1
USING BIT-FIELDS STRUCTURE

```
#define TYPE_MANAGEMENT 0
#define MGM_SUBTYPE_BEACON 8

typedef struct FrameControl_t {
    uint8_t ProtocolVersion :2;
    uint8_t Type :2;
    uint8_t Subtype :4;
    // ... other fields ...
} FrameControl_t;


// check the Type/Suptye field of frame
FrameControl_t *fc = &frame->FrameControl;

if (fc->Type != TYPE_MANAGEMENT ||
    fc->Subtype != MGM_SUBTYPE_BEACON) {
    // discard the frame...
}
// parse beacon frame...
```

```
network SSID: UniversityStudent
      BSSID             RSSI    Channels
38:1C:1A:79:0F:18       -70     1
38:1C:1A:C4:35:F8       -78     1
58:97:1E:A2:60:B8       -81     1
00:08:2F:32:F3:D8       -49     1
00:08:2F:4A:90:28       -67     6
BC:67:1C:E8:FF:48       -79     6
BC:67:1C:E8:FB:68       -82     6
6C:FA:89:C9:47:18       -57     11
38:1C:1A:C4:31:28       -76     11
```

Figure 9. Captured information about UniversityStudent network

VII. Performance evaluation

In this section, we proceed with the power measurement to find out if passive scan brings a power saving as soon as it uses only the receiver which consumes less energy than the transmitter, which is used in the active scan.

A. Prerequisites

To make a fair comparison of power consumption between the default active scan and the implemented passive scan, we had to restrict some functionality of a passive scan and set similar parameters to both programs. Thus, the compared programs print a list of up to 20 entries with BSSID, RSSI, and SSID obtained during one second.

The measurement was performed with J6 jumper on the BoosterPack board. The measurement procedure is described in the hardware User's guide [8]. To record the voltage changes we used the digital oscillograph *Rigol DS1042CD*. And to measure the amperage we used digital multimeter *EM3058*. At the same time, the multimeter plays the role of the resistor required for voltage measuring (1.259Ω).

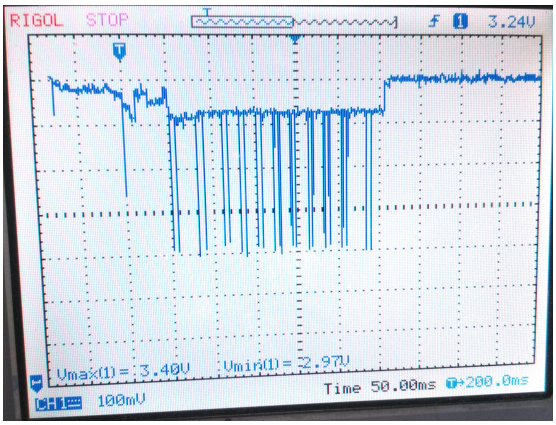

Figure 10. Active scan voltage

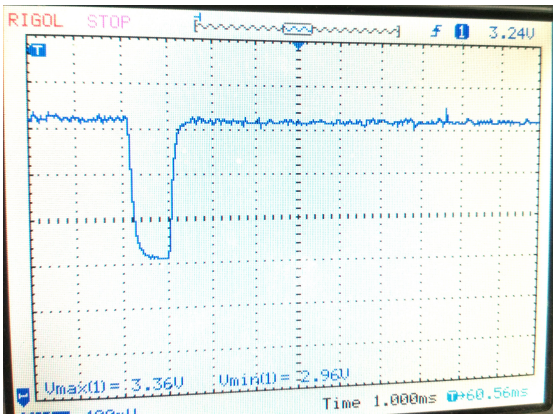

Figure 11. Voltage of one transmission pulse during active scan

B. Results

The captured active scan voltage is displayed in the figure 10. According to the scale, we observed that the scan worked during $250ms$ and there were pulses which corresponded to the working transmitter and sending probe requests. In the figure 11 one pulse is displayed in bigger scale (1ms in the square). Thus, the approximated power consumption of the transceiver is 1056mW. The rest of the time, the device was working as a receiver and its power consumption was 237.6mW. The transmission of data took about 10% of active scan time, so the consumption of active scan was 319.4mW in mean during 0.25 second.

The passive scan voltage is displayed in the figure 12. We observed that there is not any pulse that corresponds to the only working receiver as expected. Thus, the power consumption of the implemented procedure was 237.6mW during the whole second i.e. the consumption of passive scan is more efficient by consuming the $2/3$ of active scan's consumption, only if it performed during the same 0.25 seconds span. With this limitation, we can observe the scan results degradation compared with the active scan.

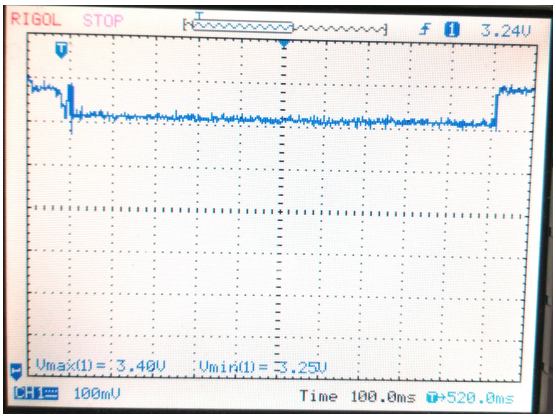

Figure 12. Passive scan voltage

VIII. Related works

- The "Passive Wi-Fi" paper[1] demonstrates the method of reaching the power saving using backscatter communication. The authors used that physic phenomenon to move the function of generating carrier wave to the plugged-in device, thereby allowing the sensors transmit 802.11b frames with consuming tens microwatts instead of hundred milliwatts as was observed in our measurement.
- The IoTScanner project[5] was made to collect and classify the wireless traffic under the assumption of passive observation of captured traffic without interfering into the work of IoT devices. They developed the program which captured and classified video-cameras traffic by its density in real time.

IX. Conclusion

We implemented the passive scan for the CC3100 microprocessor in transceiver mode. Additionally, we measured the power consumption to compare how efficient is the passive scan in comparison with the default active scan. It showed that in general active scan gives better results because of the execution speed. To reduce the passive scan consumption, a developer can tune this scanning procedure by specifying the listening channel(s) and/or increasing the interval between scan executions.

REFERENCES

[1] Bryce Kellogg, Vamsi Talla, Shyamnath Gollakota, and Joshua R. Smith. Passive wi-fi: Bringing low power to wi-fi transmissions. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 151–164, Santa Clara, CA, 2016. USENIX Association.

[2] Ritchie Dennis M. Kernighan, Brian W. *The C Programming Language*. Prentice Hall Professional Technical Reference, 2nd edition, 1988.

[3] The Institute of Electrical and Inc. Electronics Engineers. Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534, Dec 2016.

[4] Radiotap header - introduction. http://www.radiotap.org/.

[5] Sandra Siby, Rajib Ranjan Maiti, and Nils Ole Tippenhauer. Iotscanner: Detecting and classifying privacy threats in iot neighborhoods. *CoRR*, abs/1701.05007, 2017.

[6] Advanced emulation kit for simplelink™ wi-fi® cc31xx boosterpack™ plug-in module. http://www.ti.com/tool/CC31XXEMUBOOST.

[7] Cc3100 scan policies application. http://processors.wiki.ti.com/index.php/CC3100_Scan_Policies_Application.

[8] Cc3100 simplelink™ wi-fi® and iot solution boosterpack hardware. user's guide. http://www.ti.com/lit/ug/swru371b/swru371b.pdf.

[9] Cc3100/cc3200 simplelink wi-fi internet-on-a-chip. user's guide. http://www.ti.com/lit/ug/swru368b/swru368b.pdf.

[10] Simplelink wi-fi® network processor, internet-of-things solution for mcu applications. http://www.ti.com/product/CC3100.

[11] Simplelink™ wi-fi® cc3100 wireless network processor boosterpack™ plug-in module. http://www.ti.com/tool/CC3100BOOST.

[12] Libpcap file format - wireshark wiki. https://wiki.wireshark.org/Development/LibpcapFileFormat.

APPENDIX A
CAPTURED BEACON FRAME

Here is an example of a captured beacon frame visualized with WireShark.

The section one in the picture represents the general header of the management frame. We see the Frame Control field, which contains information that specifies it as a beacon frame. After duration field we see three addresses: Receiver or Destination address which is represented as the broadcast address (`FF:FF:FF:FF:FF:FF`), Destination or Transmitter address which is represented as address of our Asus Access Point (`BC:EE:7B:EC:6F:18`) and the Basic Service Set Id which is the same with MAC address of the AP in this case.

Section two represents the fields which AP should put first: Timestamp, Beacon Interval with default 1024 millisecond and capabilities information. After that, in the third section, WireShark shows the list of *elements*, which is called "Tagged parameters". Among these parameters there is SSID field with the name of our network – "CC3100SNE", supported rates from 1 to 54 Mbps which correspond to 802.11b and current working channel – 10, which we pre-configured in the AP control panel.

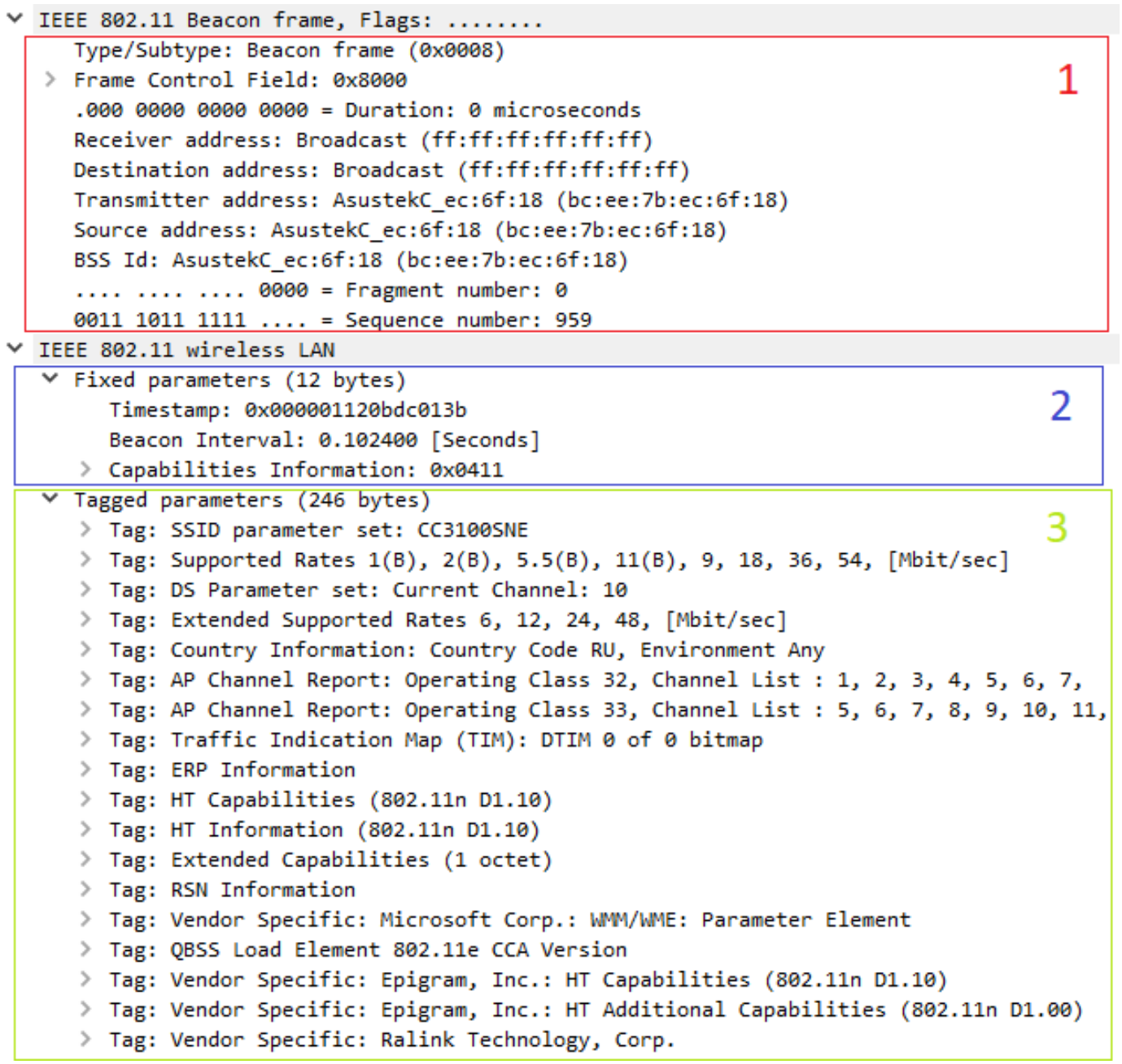

Figure 13. Captured beacon frame by CC3100