



**UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA**

Sniffing experiments with different HW devices and packets analysis

Students: Omar Moukawim, Federico Tiberti, Ferdinando Rosella

Course: Embedded Systems

Professors: Luigi Pomante, Marco Santic

Contents

1	Introduction	1
1.1	BLE spectrum	1
1.2	Report organization	2
2	Nordic HW configuration	3
2.1	nRF Connect Programmer	3
2.2	nRF Sniffer for Bluetooth LE	5
3	Wireshark usage for BLE sniffing	6
3.1	Setup of Wireshark to Display Sniffed Packets	6
3.2	Running the nRF Sniffer	9
3.3	Wireshark data dissection	10
4	PacketSniffer from Texas Instruments	12
4.1	xi_ti-psd2txt.py analysis	13
4.2	Documentation	16
4.3	Code output	19
5	Raspberry real time sniffing	20
5.1	Raspberry Setup	20
5.2	ffSniffer.py	24
6	Conclusion	27
6.1	List of files	27
	List of Figures	28

Chapter 1

Introduction

In this project the aim is to analyze Bluetooth Low Energy (BLE) packets. In general the practice of sniffing is very popular and useful for many aspects, but, first of all: what is it?

A packet sniffer is a computer program or computer hardware such as a packet capture appliance, that can intercept and log traffic that passes over a computer network or part of a network. Packet capture is the process of intercepting and logging traffic. As data streams flow across the network, the analyzer captures each packet and, if needed, decodes the packet's raw data, showing the values of various fields in the packet.

1.1 BLE spectrum

The spectrum of the BLE technology is the following one:

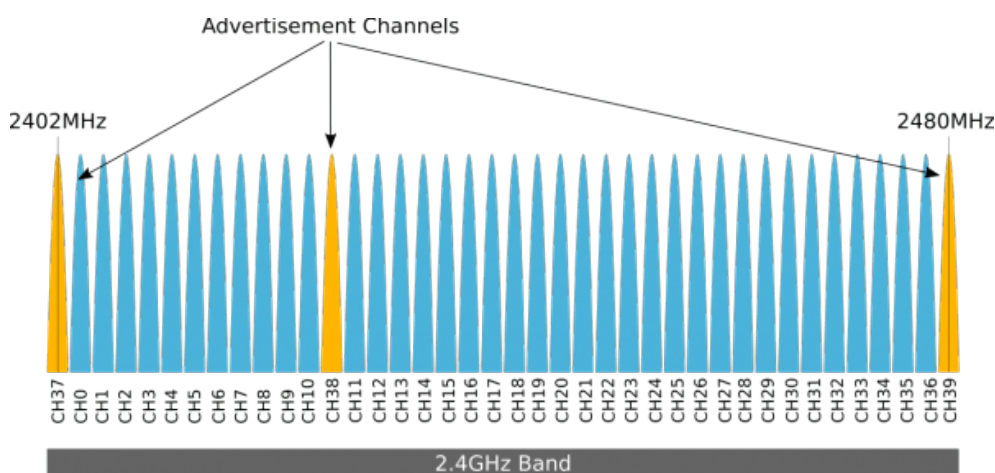


Figure 1.1: BLE spectrum

The yellow channels have been used for the sniffing activity, that can be done either taking them one at a time or taking them all together. It depends on the sw tool used, more details in the next section.

1.2 Report organization

The report will be organized in this way:

1. The second chapter to explain the work done with Nordic SemiConductor hw devices (Development Kit nRF52 and Dongle nRF52840).
2. The third chapter to show the Wireshark usage, in which all the three advertising channels have been sniffed in one time.
3. The fourth chapter will illustrate the Texas Instruments sniffing environment, composed by a PacketSniffer tool and an hw device (Dongle CC2540).
In this case, the tool can only sniff an advertising channel at a time. Moreover, in this chapter will be also presented a Python script to analyze and decompose the recorded packets by the Texas tool.
4. The fifth chapter is dedicated to Raspberry-Pi Sniffing based on the Texas Dongle CC2540.
5. The last chapter is dedicated to conclusions.

Chapter 2

Nordic HW configuration

In this chapter the setup of the Nordic HW is going to be explained step by step starting from the installation of the adequate SW and the later programming of the actual Development Kit.

2.1 nRF Connect Programmer

nRF Connect Programmer is an application available from the nRF Connect for Desktop that you can use to program firmware to Nordic devices. The application allows you to see the memory layout for both J-Link and Nordic USB devices. It also allows you to display content for the HEX files and write it to the devices.

Installation of the Programmer app

The Programmer app is installed as an app for nRF Connect for Desktop. Before you can install the Programmer app, you must download and install nRF Connect for Desktop To install the Programmer app:

1. Open nRF Connect for Desktop.
2. Find the Programmer app in the list of apps and click Install.

Once the app is installed, you can launch it by clicking Open.

nRF Connect Programmer overview

The nRF Connect Programmer main window shows the memory layout of device and file you want to work with. It also provides you with options to program. When you start the Programmer app, the following main window appears:

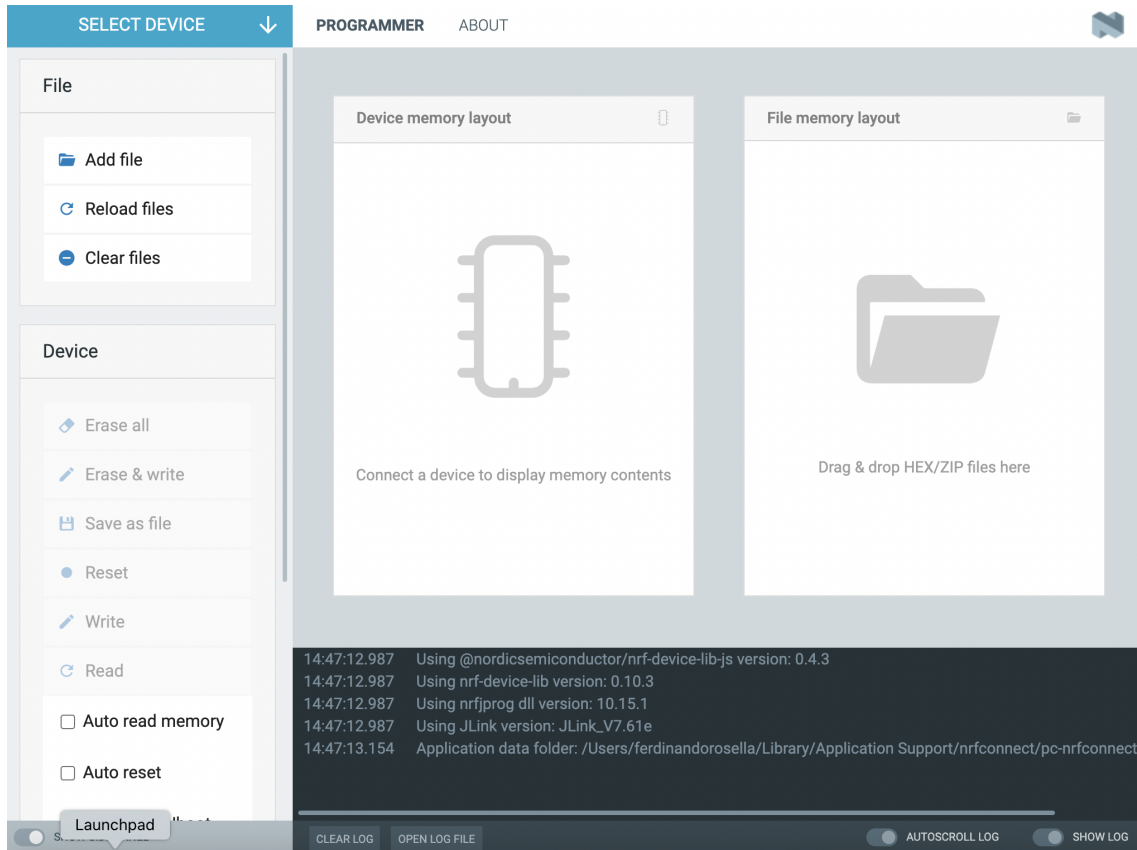


Figure 2.1: nRF Connect Programmer default view at startup

In the navigation bar at the top, you can access the menu, select a device, and see the connection status of the selected device. You can choose a device from the list of connected devices to perform further actions on the device such as programming. In our case we have programmed our devices as sniffers. The indicator is green when the Programmer app has established a connection to the device. When you select a device, you have the following actions available in the Device section:

- **Erase all** clears the written memory on the device.
- **Read** reads and displays the written memory in the Device Memory Layout.
- **Erase and write** clears the written memory and programs the files added to the File Memory Layout, this option is possible only after loading a memory layout.
- **Reset** resets the device.
- **Write** programs the files added to the File Memory Layout.

In the **File section**, you can add files to the File Memory Layout graphic, reload, and remove them. When adding files with the **Add HEX file** button, you can select the files either from the drop-down list of previous files or by browsing to the file destination.

2.2 nRF Sniffer for Bluetooth LE

The nRF Sniffer for Bluetooth LE provides a near real-time display of Bluetooth packets that are sent between a selected Bluetooth Low Energy device and the device it is communicating with. When developing a Bluetooth Low Energy product, knowing what happens over-the-air between devices can help you identify and fix issues quickly. On startup, the nRF Sniffer lists all nearby Bluetooth Low Energy devices that are advertising, providing the Bluetooth address and address type, complete or shortened name, and Received Signal Strength Indication (RSSI).

Before starting, we have checked to have the required hardware and software:

- Windows 10, 64-bit OS X/macOS 10.6 or Linux
- Wireshark v3.4.7 or later
- Python v3.6 or later
- nRF52840 Dongle (PCA10059)
- nRF52 DK (PCA10040)

Installing and Programming the nRF Sniffer

The nRF Sniffer for Bluetooth LE software consists of firmware that is programmed onto a Development Kit (DK) or dongle and a capture plugin for Wireshark that records and analyzes the detected data. After having downloaded **nRF Sniffer for Bluetooth LE v4.x or later** from [here](#) (in particular the folder: `nrf_sniffer_for_bluetooth_le_4.1.0.zip`) and having extracted it into a folder, we are able to program the **DK or dongle**, completing the following steps:

1. Install nRF Connect Programmer (as seen in section 2.1).
2. On macOS and Linux, install the **SEGGER J-Link** software but on Windows, the J-Link software is included in nRF Connect for Desktop, so it doesn't have to be installed.
3. Locate the firmware **HEX** file for our DK or dongle (located in `Sniffer Software/hex/`) and program them with the nRF Programmer tool in the nRF Connect for Desktop application.

Chapter 3

Wireshark usage for BLE sniffing

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development. Wireshark is cross-platform, using the Qt widget toolkit in current releases to implement its user interface, and using pcap to capture packets.

3.1 Setup of Wireshark to Display Sniffed Packets

The nRF Sniffer for Bluetooth LE software is installed as an external capture plugin in Wireshark. To install the nRF Sniffer capture tool on Wireshark, complete the following steps:

1. Install the python requirements:

- Open a command window in the *Sniffer-Software/extcap/folder*.
- Install the Python dependencies listed in *requirements.txt* in the folder downloaded in the previous chapter.

2. Copy the nRF Sniffer capture tool into Wireshark's folder for personal external capture plugins:

- Go to **Help** > About **Wireshark** or Wireshark > About Wireshark.
- Double-click the location for the **Personal Extcap** path that opens a folder.

CHAPTER 3. WIRESHARK USAGE FOR BLE SNIFFING

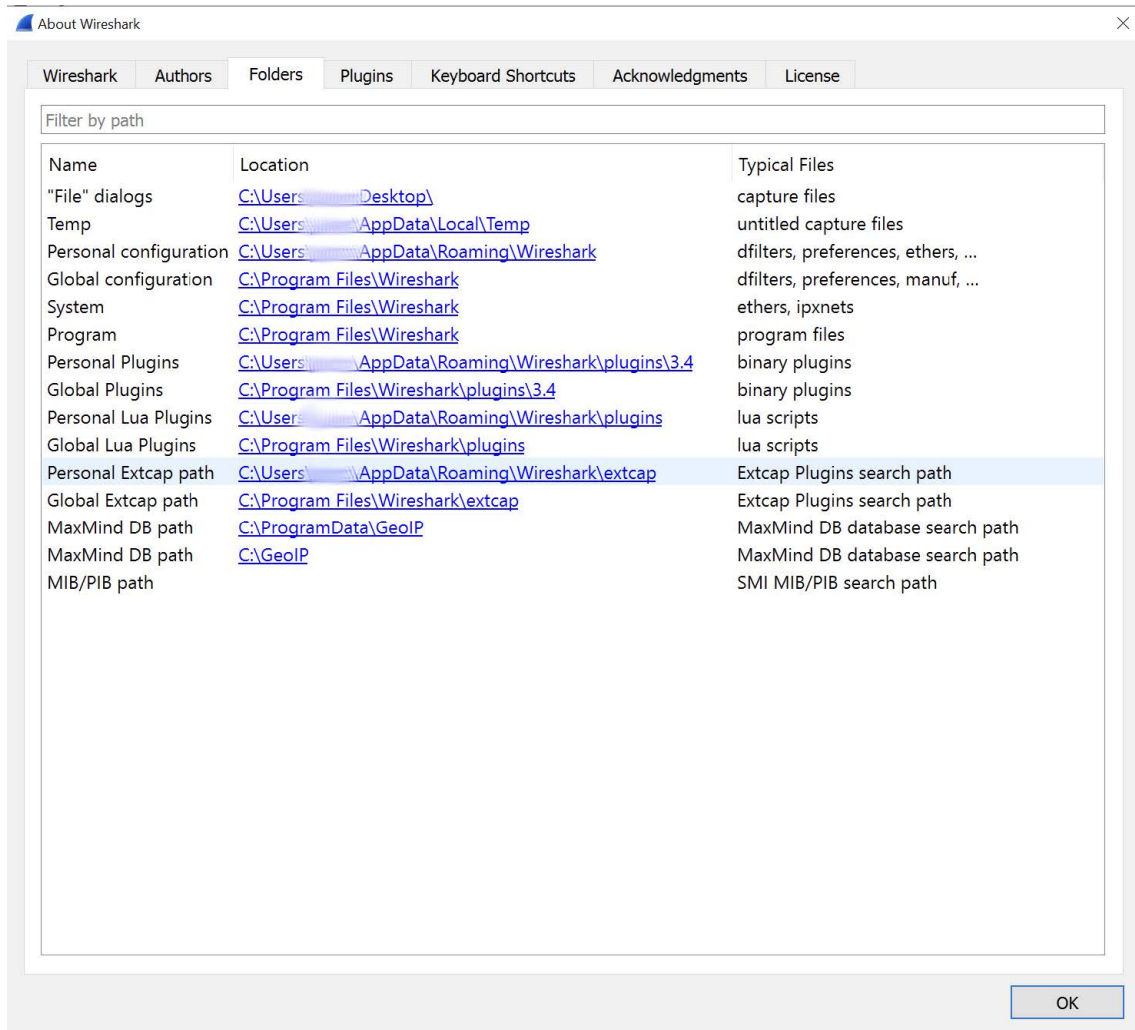


Figure 3.1: About Wireshark

- Copy the contents of the *Sniffer_Software/extcap/* into *AppData > Roaming > Wireshark > extcap*.
3. Enable the nRF Sniffer capture tool in Wireshark:
- Refresh the interfaces in Wireshark by selecting **Capture > Refresh Interfaces** or pressing **F5**.
 - Select **View > Interface Toolbars > nRF Sniffer for Bluetooth LE** to enable the nRF Sniffer interface.

CHAPTER 3. WIRESHARK USAGE FOR BLE SNIFFING

When everything is set up Wireshark should look like the following picture:

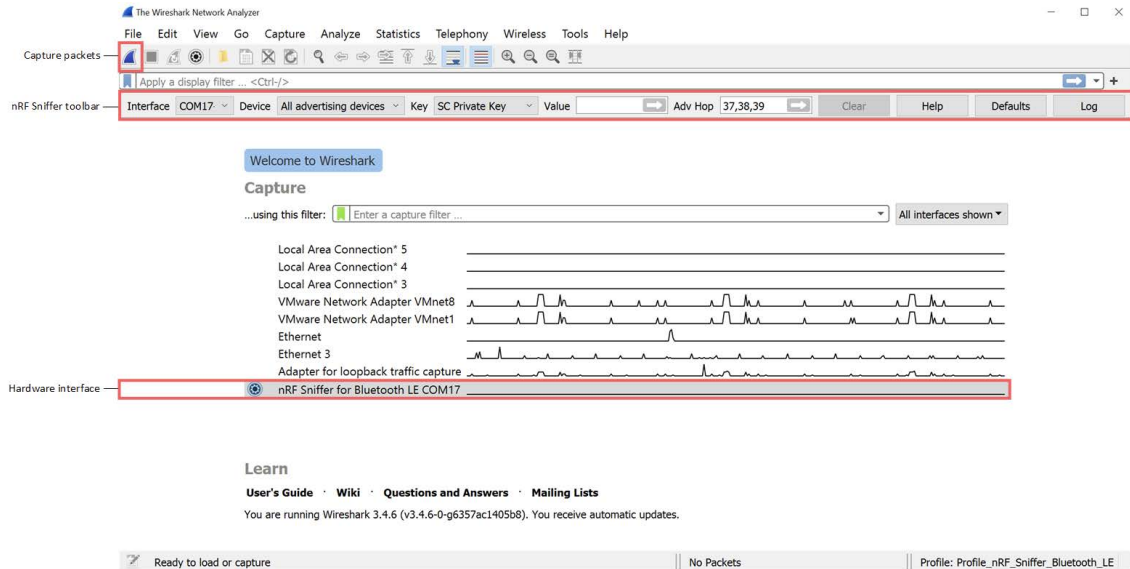


Figure 3.2: Wireshark Interface ready for sniffing

Finally for displaying the data recorded by the nRF sniffer for Bluetooth LE we have to add an nRF Sniffer Profile in Wireshark following these steps:

- Go to **Help > About Wireshark** or **Wireshark > About Wireshark**.
- Select the **Folders** tab.
- Double-click the location for the **Personal configuration** in fig 3.1.
- Copy the profile folder *Sniffer_Software/Profile_nRF_Sniffer_Bluetooth_LE* into the profiles subfolder into AppData > Roaming > Wireshark > profiles.
- In Wireshark, select **Edit > Configuration Profiles**.
- Select **Profile_nRF_Sniffer_Bluetooth_LE** and click OK as in the following figure

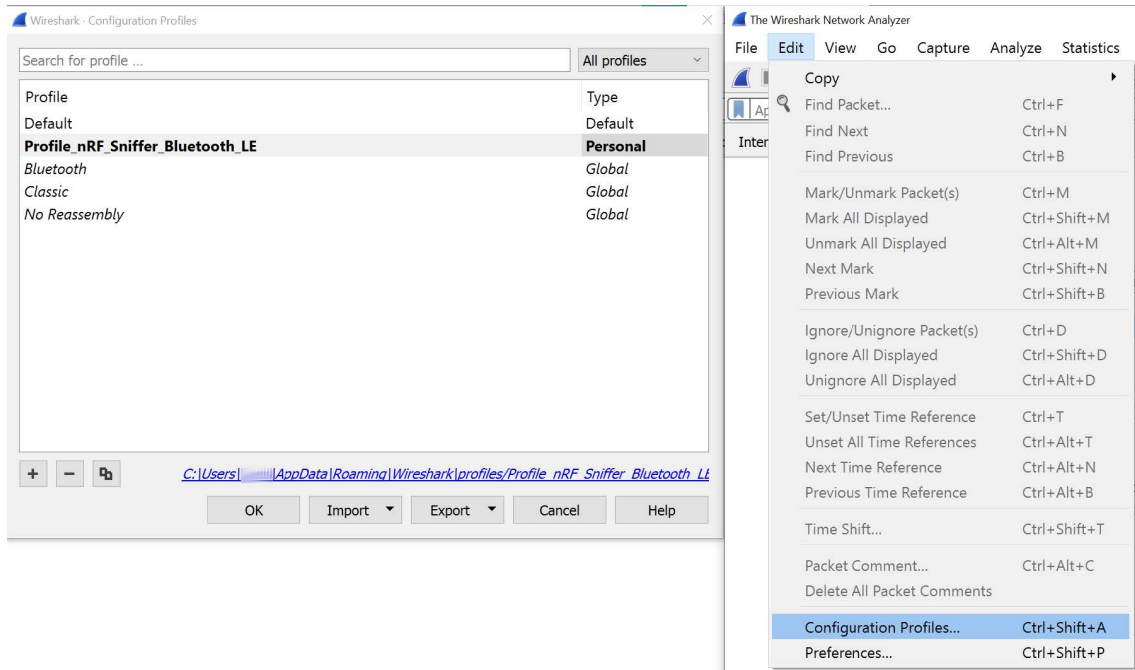


Figure 3.3: Configuration Properties

3.2 Running the nRF Sniffer

To start sniffing, place the DK or dongle that runs the nRF Sniffer for Bluetooth LE firmware between the two devices that are communicating. Then open Wireshark and start recording packets. Connect the DK or dongle to your computer and turn it on. Then place it between the Central and Peripheral device that you want to sniff.

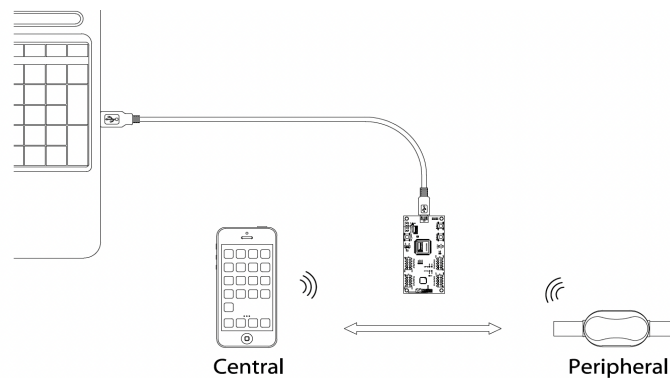


Figure 3.4: Hardware Setup

The following options and many more are available from the capture screen in Wireshark fig.3.2:

- Interface Options
- RSSI Filter
- Only advertising packets

Once the nRF Sniffer for Bluetooth LE is running, it reports advertisements and lists nearby devices in the Device List. The software interface has several commands for controlling the operating mode of the nRF Sniffer. The nRF Sniffer has two modes of operation:

1. Listen on all advertising channels to pick up as many packets as possible from as many devices as possible. This is the default mode.
2. Follow one particular device and try to catch all packets sent to or from this particular device. This mode catches all:
 - Advertisements and Scan Responses sent from the device
 - Scan Requests and Connect Requests sent to the device
 - Packets in the connection sent between the two devices in the connection

3.3 Wireshark data dissection

All the Bluetooth Low Energy packets detected by the Sniffer for Bluetooth LE are passed to Wireshark, where they are wrapped in a header containing useful meta-information not present in the Bluetooth Low Energy packet itself. Wireshark dissects the packets and separates the actual packet from the meta-information.

When browsing captured packets, selecting a packet in the packet list shows the breakdown of that packet in the packet details pane. The bytes of the packet are shown in the packet bytes panel. What said above is represented in the following picture:

Chapter 4

PacketSniffer from Texas Instruments

The main focus of this chapter will be to illustrate a Python script to decompose the sniffed packets from the tool **PacketSniffer**, using the **CC2540 dongle**. Both the software tool and the hardware device are provided by Texas Instruments.

The reason why we are going to use a Python script is to convert the output file of the software tool, that is in psd format, in a txt file.

The following figure shows a typical view inside the SW tool:

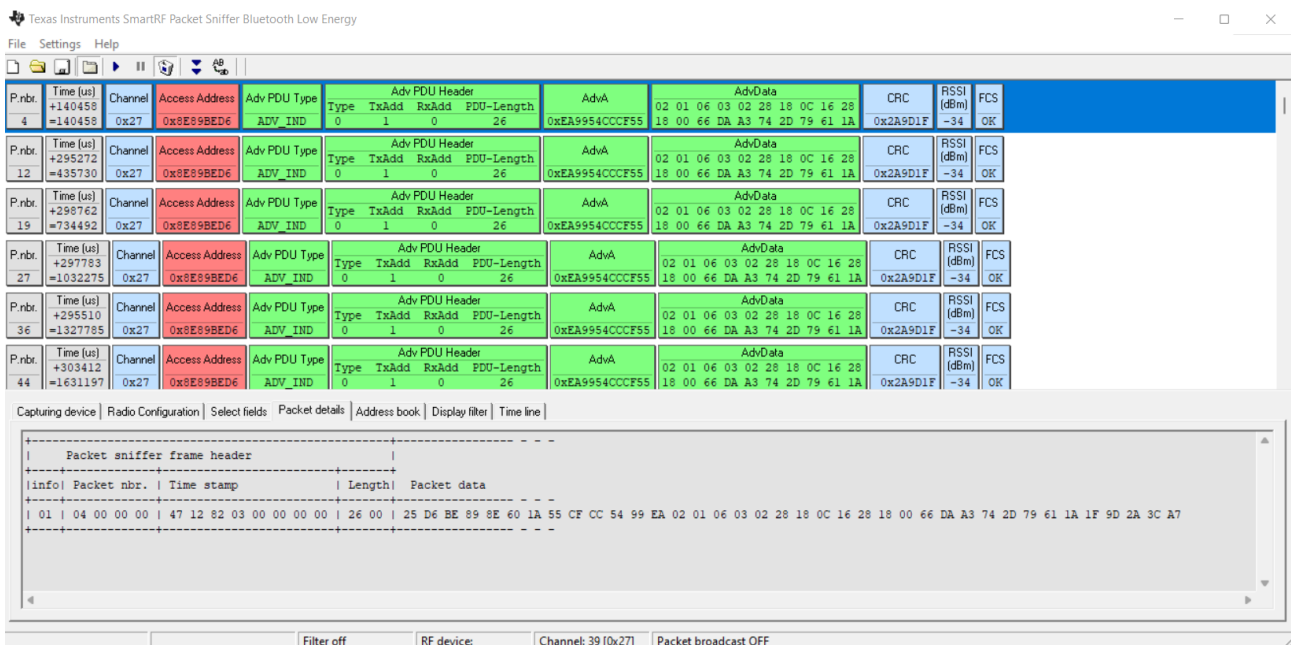


Figure 4.1: Example of a sniffing result

From the image, it is visible the packet structure and it is easy to understand that the field "packet data" inside the "packet details" tab contains several informations that are displayed above. What is before the packet data is the PacketSniffer header, added by Texas. All the sub-blocks of the packet will be analyzed with the code in the next section.

4.1 xi_ti-psd2txt.py analysis

Given a file (test1_F55.psd saved from PacketSniffer, in this case), the first thing that the code does is to open in reading/binary mode, store and close it. After that, a list (packets) is created by adding row by row the read packets.

This is done by the following lines:

```
TIRECLEN = 271
capfile = open("test1_F55.psd", "rb")
capturedata = capfile.read()
capfile.close()
for i in range(0, len(capturedata), TIRECLEN)
    packets.append(capturedata[i:i+TIRECLEN])
```

Where TIRECLEN is a constant, that is the maximum length of one packet, based on the variable size of the payload and "packets" is an empty list. Then, a for cycle starts to scan every element (packet) of the list "packets".

```
for packet in packets:
    # first unpacking - Texas header
    (pinfo, pnum, pts, plen) = struct.unpack('<ciQh', packet[0:
                                                15])

    # c - char = 8 bit (1 byte) -> pinfo
    # i - int = 4 byte -> packet number
    # Q - unsigned long long (int) = 8 bytes -> timestamp
    # h - short (int) = 2 bytes -> packet length

    # for Format Characters of struct unpack, info here:
    # https://docs.python.org/3/library/struct.html#struct.
    #      unpack

    print("pinfo = {}".format(pinfo))
    print("pnum = {}".format(pnum))
    print("pts (timestamp-decimal) = {}".format(pts))
    print("plen = {}".format(plen))

    if pinfo != b'\x01': continue
    # dovrebbero essere tutti 0x01 (adv. packets)

    # base timestamp
    if pnum == 1: tbase = pts

    data = packet[15:15 + plen] #packet data
    payload = data[0:-2]
```

```

print("data (hex) = {}".format(binascii.hexlify(data)))
print("payload (hex) = {}".format(binascii.hexlify(payload)
    ))

rssi_fcs = data[-2:] # last two elements of the data

# the first one is the rssi, while the second is divided
# into:
# [ - - - - - ] (0-6) 7 bit for the channel number
# last bit for CRC OK or NOT OK

print("rssi_fcs (hex) = {}".format(binascii.hexlify(
    rssi_fcs)))

rssi = rssi_fcs[0] - 94
exflags = rssi_fcs[1] # last byte

print("rssi = {}".format(rssi))

channel = exflags & 0x7f # masking with 0111 1111 (bit 0-6
    for channel number)
fcsok = (exflags & 0x80 > 0) # masking with 1000 0000 (bit
    7 for crc ok or not)

print("channel number = {}".format(channel))
print("CRC OK = {}".format(fcsok))

# last 2 bytes of the data have been analyzed.

# let's start with the payload decomposition:

(ble_pre, ble_aa) = struct.unpack('< cI', payload[0:5])
ble_pdu = payload[5:-3]
# il ble_pdu has 2 byte for BLE HEADER and the others are
# the payload

ble_crc = payload[-3:]
ble_pre = hex(int.from_bytes(ble_pre, byteorder='big')) #
    FROM BYTE TO INT AND AFTER
    FROM INT TO HEX

print("ble_pre (hex) = ", ble_pre)
print("acc_addr (decimal) = {}".format(ble_aa)) # IN
    DECIMAL
print("ble_pdu (hex) = {}".format(binascii.hexlify(ble_pdu)
    ))
print("ble_crc (hex) = {}".format(binascii.hexlify(ble_crc)
    ))

pdu_head = ble_pdu[0:2] # first 2 payload bytes are for
    BLE HEADER
pdu_payload = ble_pdu[2:]

print("pdu_header (hex) = {}".format(binascii.hexlify(
    pdu_head)))

```



```

print("pdu_payload (hex) = {}".format(binascii.hexlify(
    pdu_payload)))

pdu_pl_len0 = len(pdu_payload)
print("pdu_pl_len0", pdu_pl_len0)
'''
FURTHER ANALYSIS ON DATA PACKETS

DOCUMENTATION HERE: https://www.mdpi.com/1424-8220/12/9/11734

ARTICLE:
Gomez, Carles, Joaquim Oller, and Josep Paradells.
"Overview and evaluation of bluetooth low energy: An emerging
    low-power wireless technology
    ."
Sensors 12.9 (2012): 11734-11753.

MIC = pdu_payload[-4:] #Message Integrity Check (4 bytes)
L2HE = pdu_payload[0:4] #L2CAP HEADER (4 bytes)
Op = pdu_payload[5] #ATT Opcode (1 byte)
par_pay = pdu_payload[5:-4] #parameters and payload <= 22
    bytes
print("----- par/pay length: ", len(par_pay))
'''

# ADVERTISING PACKETS ANALYSIS (OUR INTEREST):
# info here: https://www.bluetooth.com/blog/bluetooth-low-energy-it-starts-with-advertising/

# 2 BYTE HEADER decomposition:
# RxAdd (1 bit)|TxAdd (1 bit)|RFU (2 bit)|Tipo PDU (4 bit)|
    RFU (2 bit)|Lunghezza (6
    bit)

RXrnd = ((pdu_head[0] & 0x80) > 0)
TXrnd = ((pdu_head[0] & 0x40) > 0)
pdu_type = pdu_head[0] & 0x0F
pdu_pl_len = pdu_head[1] & 0x3F
print("TXrnd", TXrnd)
print("RXrnd", RXrnd)
print("pdu_type = {}".format(pdu_type))
print("pdu_pl_len = {}".format(pdu_pl_len))

if (pdu_pl_len0 != pdu_pl_len): continue

adv_addr = pdu_payload[0:6]
adv_data = pdu_payload[6:]
print("adv_addr (MAC address on nRF CONNECT APP) = 0x",
    ''.join('{:02X}'.format(x) for x in adv_addr[:-1]))
    # for che legge
    tutta la lista al
    contrario
print("adv_data = 0x", ''.join('{:02X}'.format(x) for x in
    adv_data))

```

The main references used to create the code are:

1. SmartRF™ Packet Sniffer User's Manual
For basic packets analysis
2. For further payload decomposition [here](#)

Further details are reported in the next paragraph.

4.2 Documentation

From the Packet Sniffer User's Manual, the following informations are provided:



4 Format of packets saved to file

The figure below describes the packet format for packets saved to a Packet Sniffer Data (PSD) file. The number of bytes is given for each field.

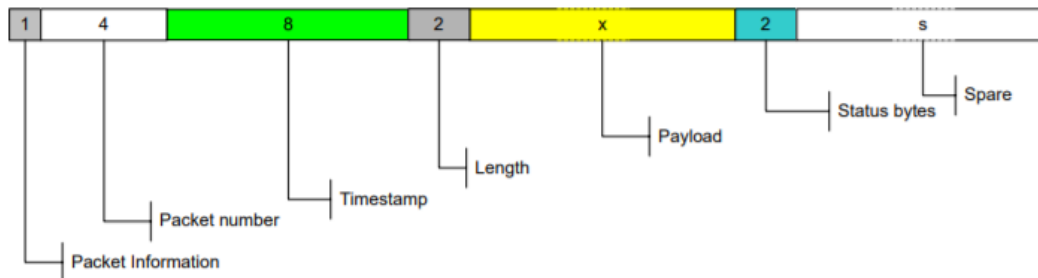
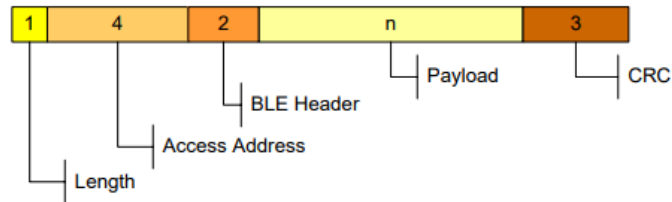


Figure 4.2: Packets format

Where the BLE payload, the status bytes and the spare bytes are:

BLE devices:



The complete packet as sent over the air. For more details see the BLE protocol specification.

Status bytes:

The Status bytes are generated by the RF Device used as capturing device. See the user guide of the capturing device for more details.

BYTE 1: RSSI.
 BYTE 2: Bit 7: Indicate CRC OK or not.
 Bit 0-6: Depending on the RF Device and the configuration of these.

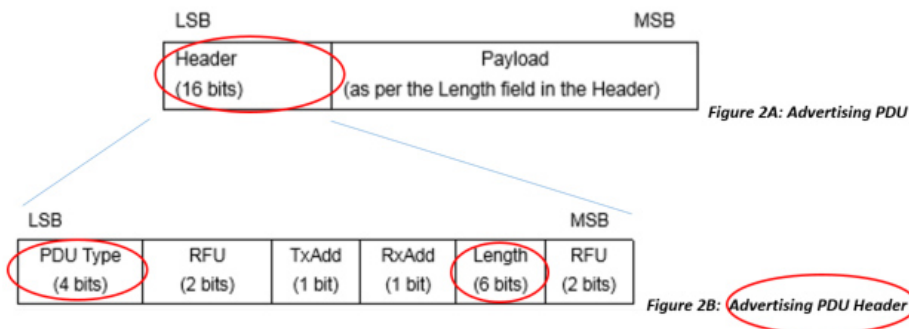
Note: For BLE devices bit 0-6 in byte 2 is set to the channel number by the packet sniffer FW.

Spare:

The number of spare bytes depends on the total amount of bytes used by the packet sniffer to save the packet. The number of bytes depends on the protocol and can be seen from the description of the packet format under the help menu.

Figure 4.3: Packets details

Then, further analysis have been done on the payload (light yellow in Figure 2.3). Referring to the Bluetooth link, this is the structure:



Figures 2A and 2B – Advertising PDU header, specifically PDU type and Length.

Figure 4.4: Payload details

Where there are 4 types of PDU Type:

1. ADV_IND - Advertising Indication
A peripheral device requests connection to any central device (i.e., not directed at a particular central device).
Example: A smart watch requesting connection to any central device.
2. ADV_DIRECT_IND - Advertising Direct Indication
Yet the connection request is directed at a specific central device.
Example: A smart watch requesting connection to a specific central device.
3. ADV_NONCONN_IND
Non connectable devices, advertising information to any listening device.
Example: Beacons in museums defining proximity to specific exhibits.
4. ADV_SCAN_IND
Similar to ADV_NONCONN_IND, with the option additional information via scan responses.
Example: A warehouse pallet beacon allowing a central device to request additional information about the pallet.

4.3 Code output

A typical script output is the following one:

```
pinfo = b'\x01'
pnum = 4
pts (timestamp-decimal) = 58856007
plen = 38
here
data (hex) = b'25d6be898e601a55cfcc5499ea020106030228180c1628180066daa3742d79611a1f9d2a3ca7'
payload (hex) = b'25d6be898e601a55cfcc5499ea020106030228180c1628180066daa3742d79611a1f9d2a'
rssi_fcs (hex) = b'3ca7'
rssi = -34
channel number = 39
CRC OK = True
ble_pre (hex) = 0x25
acc_addr (decimal) = 2391391958
ble_pdu (hex) = b'601a55cfcc5499ea020106030228180c1628180066daa3742d79611a'
ble_crc (hex) = b'1f9d2a'
pdu_header (hex) = b'601a'
pdu_payload (hex) = b'55cfcc5499ea020106030228180c1628180066daa3742d79611a'
pdu_pl_len0 26
TXrnd True
RXrnd False
pdu_type = 0
pdu_pl_len = 26
adv_addr (MAC address on nRF CONNECT APP) = 0x EA9954CCCF55
adv_data = 0x 020106030228180C1628180066DAA3742D79611A
```

Figure 4.5: Code result

Different output formats have been chosen for two main reasons:

- For data description, because different parameters need to be expressed in different numerical bases. (e.g. RSSI has no sense in hex; pinfo has no sense in decimal; etc . . .)
- For future works, so that, data can be transformed by the user in the most suitable format for their desired applications.

Chapter 5

Raspberry real time sniffing

5.1 Raspberry Setup

As a further work, we have experienced a live sniffing using these devices:

- Raspberry-Pi
- Texas dongle, the same: CC2540
- A Silvair dongle
- iPhone - only to configure the Silvair dongle

Firstly, we have connected the Raspberry to the network and it wasn't easy, since we didn't have a physical router. To solve this issue, we have used a wireless connection storing an appropriate file within the Raspberry. The file name is: **wpa_supplicant.conf**, it stores the following code:

```
country = IT (if your country is Italy)
ctrl_interface = DIR = /var/run/wpa_supplicant GROUP = netdev
update_config = 1
network = {
scan_ssid = 1
ssid = "Your-SSID"
psk = "Your-PSK"
key_mgmt = WPA-PSK
}
```

Replacing "Your-SSID" with your WiFi network, and "Your-PSK" with your WiFi password. This file will tell the Raspberry Pi to connect to the specified network when it boots up. (i.e. when the green blink finishes)

The second step is to take the ip-address of our network, we simply read it from here:

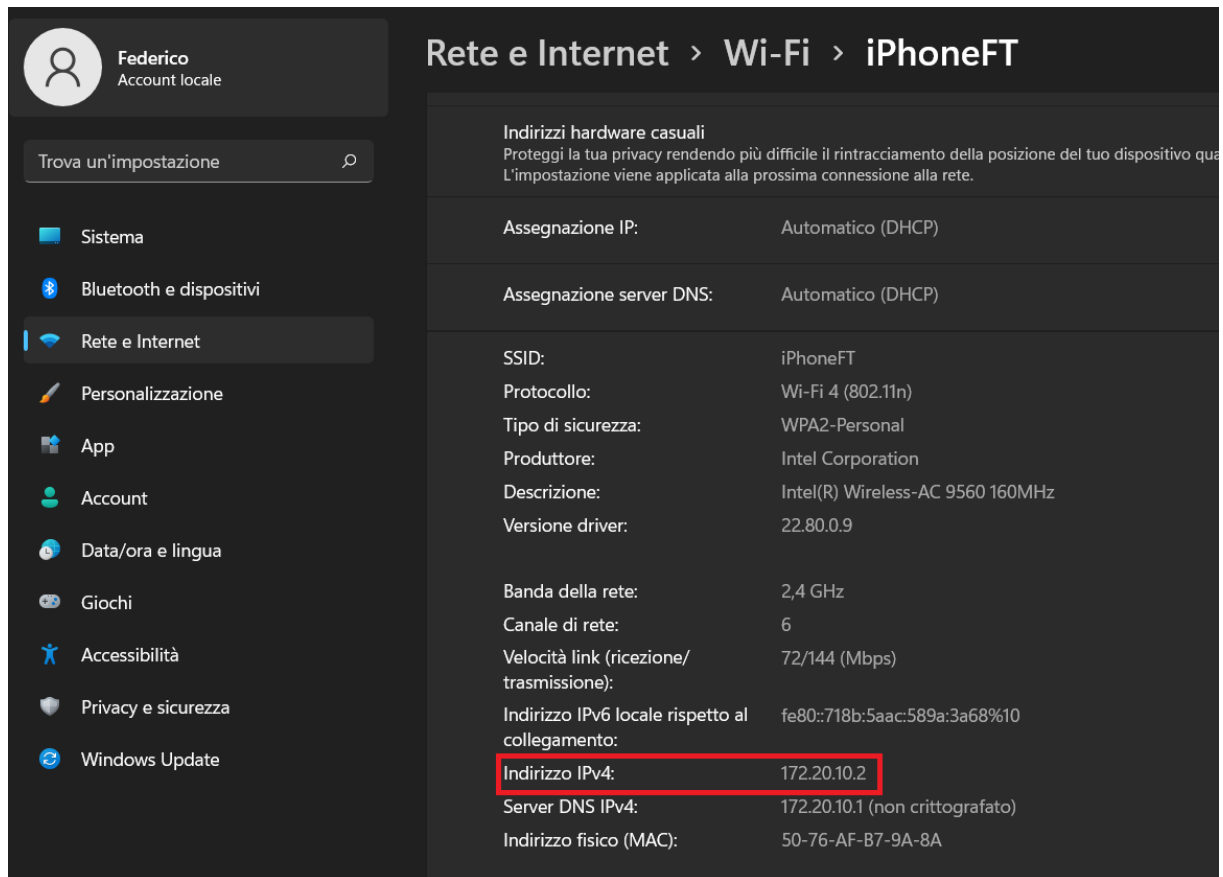


Figure 5.1: Pc IP address

Now we have the IP address of our computer, we will scan the whole subnet for other devices. For example, our IP address is 172.20.10.2, other devices will be at addresses like 172.20.10.23, 172.20.10.4, 172.20.10.1, etc. Therefore, to find the Raspberry, we have used the "nmap" software. It is available here: <https://nmap.org/download.html> for Windows, or it can be easily installed on MacOS or Linux. Now we have used the nmap command "nmap 172.20.10.0/24" to scan on the whole subnet range. The result is this:

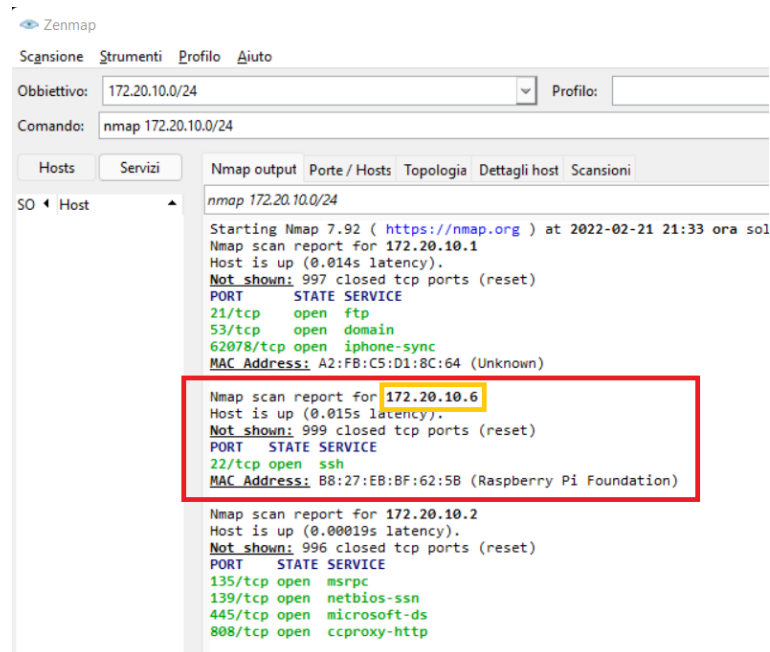


Figure 5.2: Raspberry IP address

Once we get the Raspberry IP, we can connect to it using the Putty Terminal, through a SSH connection:

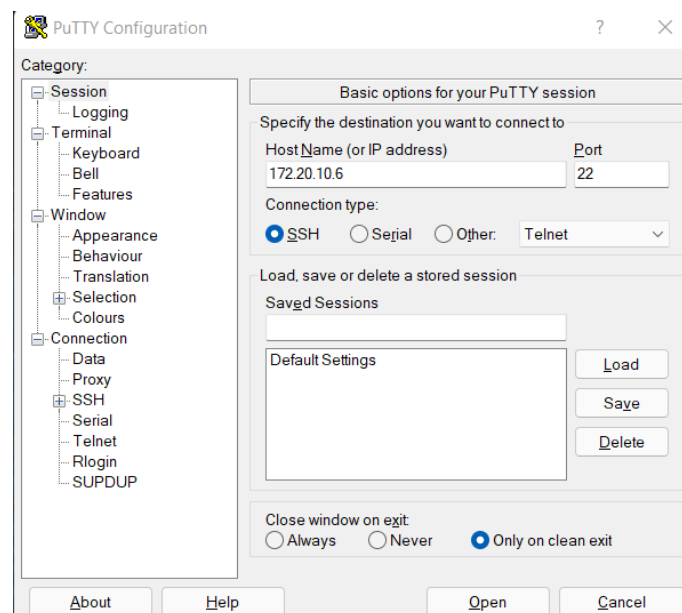
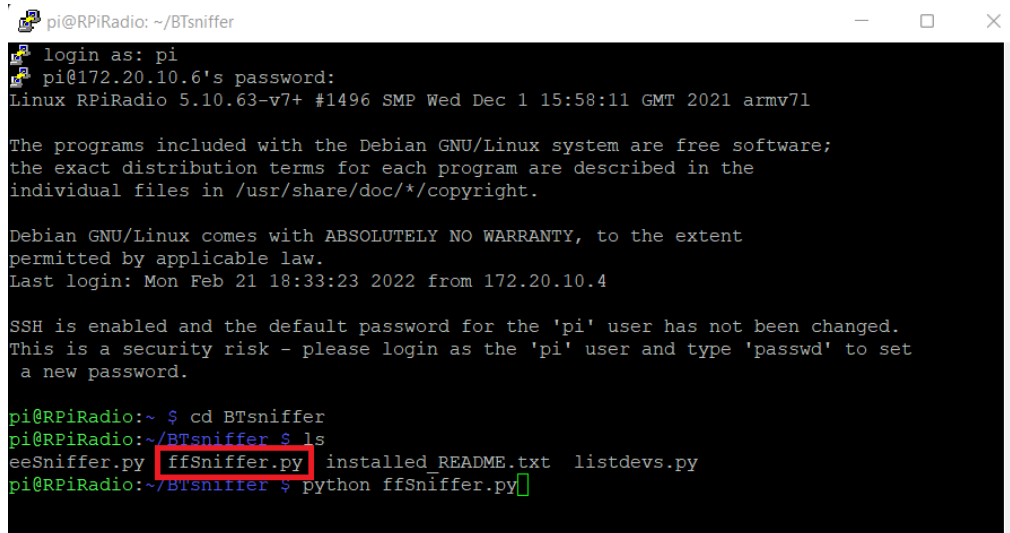


Figure 5.3: Raspberry connection

Once connected, the Pi-Terminal will be shown, login and password are

required (pi and raspberry respectively). Here an example:



```
pi@RPiRadio: ~/BTsniffer
login as: pi
pi@172.20.10.6's password:
Linux RPiRadio 5.10.63-v7+ #1496 SMP Wed Dec 1 15:58:11 GMT 2021 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Feb 21 18:33:23 2022 from 172.20.10.4

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@RPiRadio:~ $ cd BTsniffer
pi@RPiRadio:~/BTsniffer $ ls
eeSniffer.py ffSniffer.py installed README.txt listdevs.py
pi@RPiRadio:~/BTsniffer $ python ffSniffer.py
```

Figure 5.4: Raspberry terminal

The red area underlines the "ffSniffer.py" file within the BTsniffer folder. This code has been created, working on it with our PC and then uploading it to the Raspberry by means of WinSCP.

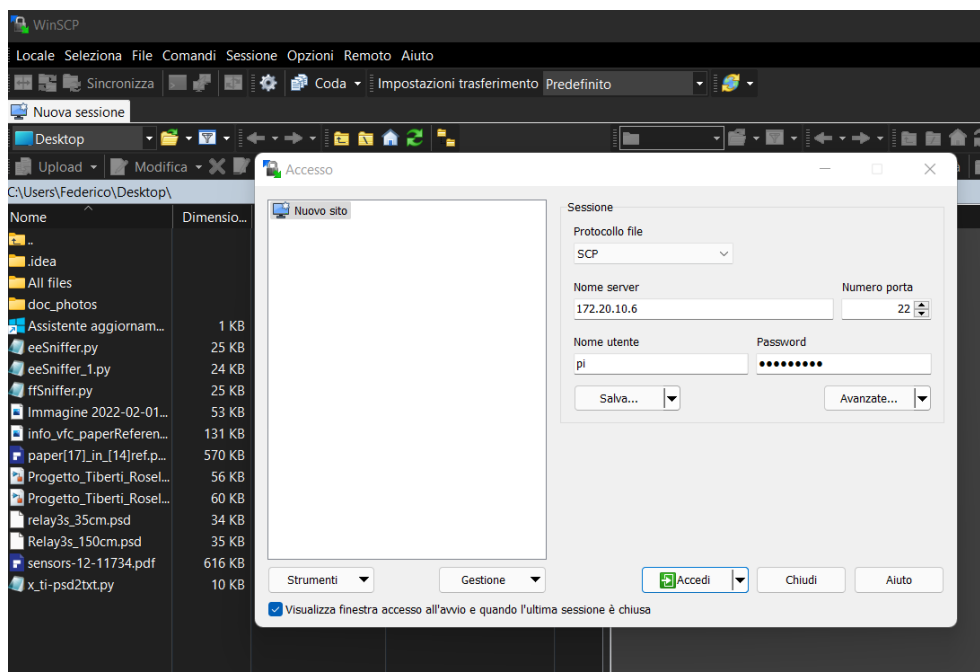


Figure 5.5: WinSCP

To correctly run the ffSniffer.py code, the following packets are required:

```

1 // pip for python3
2 sudo python -m pip install --upgrade pip
3
4 // libUSB 1
5 sudo apt-get install libusb-1.0-0-dev
6
7 // pyUSB
8 pip install pyusb
9
10 cd /etc/udev/rules.d/
11 sudo nano 99-com.rules
12 // add line at the end of file:
13 SUBSYSTEMS=="usb", ENV{DEVTYPE}=="usb_device", ATTRS{idVendor}=="0451", ATTRS{idProduct}=="16b3", GROUP="plugdev", MODE="0777"
14 // then
15 sudo udevadm control --reload
16 sudo udevadm trigger
17
18 // unplug aned replug dongle
19
20 // install six
21 pip install six
22
23 // vedere link:
24 https://github.com/pyusb/pyusb/blob/master/docs/tutorial.rst
25 https://revspace.nl/CC2540
26 https://github.com/christianpanton/ccsniffer/blob/master/ccsniffer.py
27 https://github.com/bertrik/cc2540
28 // l'applicazione eesniffer e' un mix da queste fonti
29 // l'applicazione ffSniffer è un upgrade della eeSniffer

```

Figure 5.6: Installation readMe

5.2 ffSniffer.py

The purpose of the "ffSniffer.py" code is to sniff and analyze all the packets, filtering the Silver dongle ones paying attention to correctly define the dongles addresses (add1, add2, add3, add4 fields in the code). The following are three representations of the code output: (we can choose between them)

```

PACKET NOT FROM SILVAIR

PACKET NOT FROM SILVAIR

PACKET NOT FROM SILVAIR

PACKET NOT FROM SILVAIR

PACKET FROM SILVAIR:
timestamp: 1574112011
ble_acc_addr: b'd6be898e'
ble_header: b'421e'
adv_addr: b'03160628afa5'
adv_data: b'172a6695bb6662325392b261ca1c4ad7ea2c57f4dbabd2b8'
ble_crc: b'2e2f7e'
RSSI -38
Channel 39
FCSOK True

PACKET NOT FROM SILVAIR

PACKET NOT FROM SILVAIR

PACKET NOT FROM SILVAIR

PACKET NOT FROM SILVAIR

```

```
PACKET NOT FROM SILVAIR
timestamp: 235471597
ble_acc_addr: b'd6be898e'
ble_header: b'4017'
adv_addr: b'57ed25bf3ec7'
adv_data: b'02011a020a1a0aff4c0010052718cb1f47'
ble_crc: b'0ba7fb'
RSSI -73
Channel 39
FCSOK True

PACKET FROM SILVAIR:
timestamp: 245761495
ble_acc_addr: b'd6be898e'
ble_header: b'421e'
adv_addr: b'03160628afa5'
adv_data: b'172a6604b48b1989bc52687e401ffbb6bc1343e50c403848'
ble_crc: b'177882'
RSSI -52
Channel 39
FCSOK True

PACKET NOT FROM SILVAIR
timestamp: 271125547
ble_acc_addr: b'd6be898e'
ble_header: b'4225'
adv_addr: b'0843c18fa865'
adv_data: b'1eff06000109200293766ac663a998d54f298d43e7ba72cacd4471941d73a9'
ble_crc: b'd33336'
RSSI -71
Channel 39
FCSOK True
```

```
PACKET FROM SILVAIR:
timestamp: 2842035951
ble_acc_addr: b'd6be898e'
ble_header: b'421e'
adv_addr: b'03160628afa4'
adv_data: b'172a66a2b6a6606a33aa654c67df610d6caed367061a9c31'
ble_crc: b'b707a0'
RSSI -43
Channel 39
FCSOK True

PACKET FROM SILVAIR:
timestamp: 3641966866
ble_acc_addr: b'd6be898e'
ble_header: b'421e'
adv_addr: b'03160628afa5'
adv_data: b'172b0100b35ea3be0540ce540000000f203bbeec811d884'
ble_crc: b'd6461c'
RSSI -43
Channel 39
FCSOK True

PACKET FROM SILVAIR:
timestamp: 4088137850
ble_acc_addr: b'd6be898e'
ble_header: b'421e'
adv_addr: b'03160628afa6'
adv_data: b'172a66f074b32ffe622286cb9fa696d057847e887da62c40'
ble_crc: b'ad80da'
RSSI -42
Channel 39
FCSOK True

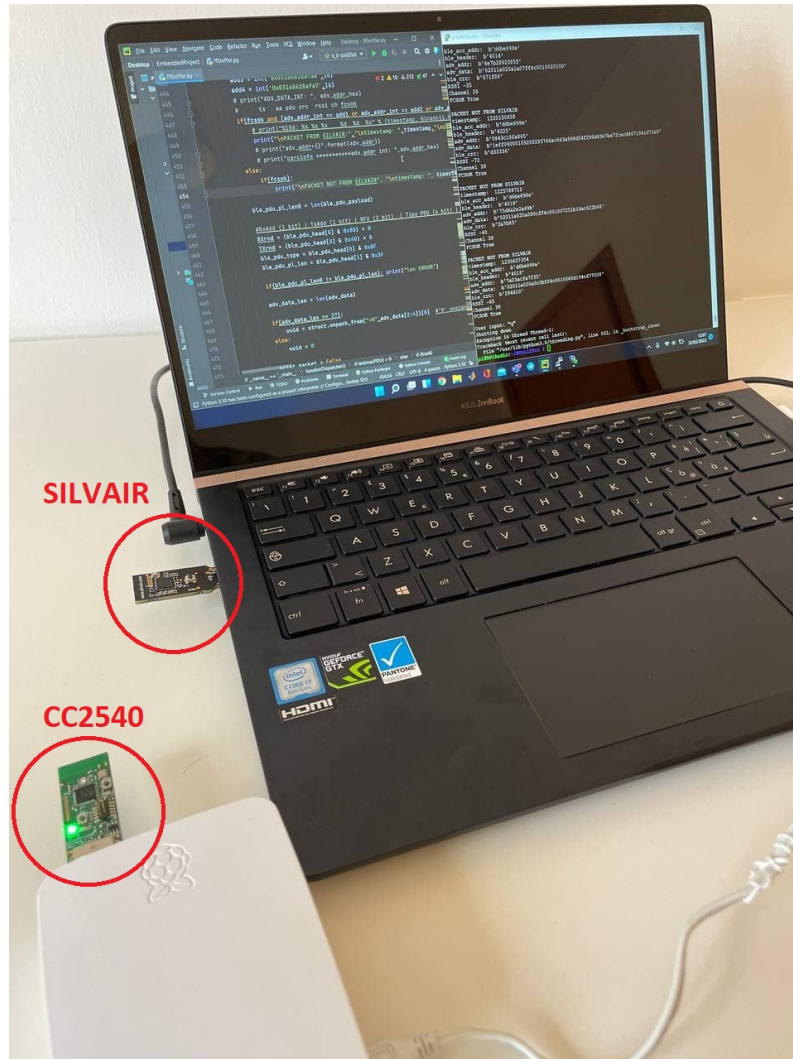
PACKET FROM SILVAIR:
timestamp: 1038813116
ble_acc_addr: b'd6be898e'
ble_header: b'421e'
adv_addr: b'03160628afa7'
adv_data: b'172a66c42136d44ba4f41575899bc74906599c4c13706e53'
ble_crc: b'13412a'
RSSI -43
Channel 39
FCSOK True
```

In the first image, we only display the Silvair dongle packet details and we print a simple "PACKET NOT FROM SILVAIR" otherwise. In the second picture we display both information about the Silvair and generic packets. Lastly, in the third, only Silvair packets are shown. This application is particularly interesting because of its real-time capability of both sniffing and dissecting packets. Instead, if you are not interested in

CHAPTER 5. RASPBERRY REAL TIME SNIFFING

real-time applications, you can refer to the code in chapter 4 to post-analyze a psd file.

As a final remark, we would like to show the practical setup of the above work:



Chapter 6

Conclusion

In conclusion we can say that this project work was a very interesting experience that allowed us to work and learn about different technologies regarding BLE under different point of views:

- The signal processing point of view (Wireless Communications)
- The physical layer point of view (Embedded Systems)

In our Wireless project the main approach regarded the analysis of RSSI (Received Signal Strength Indicator) in different environments and at different distances between the transmitter(dongle) and the receiver(sniffer).

On the other hand for the Embedded part we have analyzed the physical layer level of the packet transmission and dissected it in an appropriate manner, separating each relevant BLE field from the other (pdu, payload, ecc...).

Another interesting takeaway from this work has been learning how to correctly use a Raspberry - Pi, a skill we have never had the opportunity to practice before.

6.1 List of files

Type	Name	Chapter
.zip	nrf_sniffer_for_bluetooth_le_4.1.0.zip	2, 3
Texas	test1_F55.psd	4
Python	xi_ti-psd2txt.py	4
Python	ffSniffer.py	5

List of Figures

1.1	BLE spectrum	1
2.1	nRF Connect Programmer default view at startup	4
3.1	About Wireshark	7
3.2	Wireshark Interface ready for sniffing	8
3.3	Configuration Properties	9
3.4	Hardware Setup	9
3.5	Packet dissection	11
4.1	Example of a sniffing result	12
4.2	Packets format	16
4.3	Packets details	17
4.4	Payload details	17
4.5	Code result	19
5.1	Pc IP address	21
5.2	Raspberry IP address	22
5.3	Raspberry connection	22
5.4	Raspberry terminal	23
5.5	WinSCP	23
5.6	Installation readMe	24