Wireless internet - WiFi traffic classification.

Luca Ferraro (10748116), Fabio Losavio (10567493)
https://github.com/LucaFerraro/Wireless_internet_project

A.Y. 2019/2020

Contents

List of Figures

Listings

1 Introduction

The aim of this project is using the protocol analyzer *Wireshark* for classifying traffic at *MAC* layer.
To reach this goal, we used *Wireshark* in *monitor mode*: in this mode, the *WiFi* module of our PC intercepts all the traffic in range that is transiting via *WiFi*. This is a limitation is the sense that at *MAC* layer it's possible to check whether a frame is transporting data, but it's not possible to distinguish the application that has generated the data carried in the payload of the frame. Therefore, our analysis only provides a quantitative estimation of the traffic of data transported over *WiFi*. The reason why we have used this solution even if we were aware of this fact is that working in *monitor mode* is the only solution we had to sniff traffic that was not meant only for the computer that was performing the capure.

1.1 Our analysis.

In this project, what we have done is looking at the *data* and *QoS data* frames captured by our computers and try to understand what those packets are meant to in terms of type of application that has generated them. As specified, this can only be a deduction: we can't be sure of the application that has generated a packet at *MAC* layer.
To do this we have scanned many sample captures and for each one of them:

1. Found all the *MAC* addresses that have generated/received at least one *data/QoS data* frame.
2. Associated to all the *MAC* addresses we have revealed:
   - The number of transmitted/received bytes.
   - The number of transmitted/received packets.
   - The average uplink/downlink rate.
3. Found the vendor associated with the *MAC* address.

In order to analyze the captured traffic and try to understand the type of application a user (so a *MAC*) was using, we have plotted:

- A pair of histograms (one for the bytes, one for the number of packets) in which there is a bar for each *MAC* address that have generated/received at least a certain amount of packets (better description of this later on).
- A graph for the cumulative uplink/downlink traffic (considering then all the bytes generated/received by all the revealed *MACs*).
- A graph for the discrete traffic in uplink/downlink, that is the total amount of bytes exchanged by all the revealed *MACs* in a fixed time interval (customizable in the program).

2 Project description

2.1 Program setup

To launch and use the program, first some pre-requisites have to be meet:

- install pyshark library in order to analyze the capture file
- install numpy library to perform some numerical analysis
- install matplotlib library to visualize plots

The program can run in 3 different modes selected by adding an attribute to the standard python call, using the command `python Traffic_analyzer.py -file "PATH TO A FILE"` the program will scan the file provided after the *-file* parameter. In this mode the program will open the selected capture (in a .pcap/.pcapng format) and start scanning all the packets in order to obtain some information.

The second mode is the live capturing mode in which, using the command `sudo python Traffic_analyzer.py -live "INTERFACE" "DURATION"` the program will first start a capture on the given interface,that has to be enabled to work in monitor mode, for a given amount of time provided with the "DURATION" attribute, will save this capture and will work on it. With this mode administrator privileges are required to allow tshark to start the capture in monitor mode.

The last mode is a default mode, launched using the command `python Traffic_analyzer.py`, which will perform the analysis on a default capture inserted in the code. This mode is mostly used as a debug tool but can also be useful to understand how the output will look like.

2.2 Program structure

To obtain information from a capture, the code will run a for loop on all the packets in the capture file and will analyze only those which can be useful for our purpose. To select the useful the following filter is used:

```
1     (int(packet.wlan.fc_type) == 2) and
2         ((int(packet.wlan.fc_subtype) >= 0 and
3             int(packet.wlan.fc_subtype) <= 3)) or
4         (int(packet.wlan.fc_subtype) >= 8 and
5         int(packet.wlan.fc_subtype) <= 11)
```

Listing 1: Packet filter

it will select all the **data** frames (`wlan.fc_type == 2`) and among those it will select the ones actually containing data or QoS data excluding null packets or ACKs. From those packets it will extract the destination address and will add it to a dictionary which will store the number of bytes extracted from the `packet.data.len` and the number of packets received and transmitted by this MAC during the capture. Then it will try to extract, if present, the source

address and store information the same way. During the development phase we noticed that the source address (`wlan.packet.sa`) was not always present so we decided to check if it was present in order to avoid errors.

Before the end of the loop, the lists for the cumulative traffic curves are updated. To effectively print those curves we had to divide the time in discrete intervals of length T and check if the `packet.frame_info.time_relative` was inside the current interval. If the interval is correct, the counters are updated, otherwise a new time slot is created.

```
    # Updating traffic curve:
    if (t_capture >= n*T): # if t_capture in [(n+1)T, (n+2)T]
        traffic_in.append(nBytes_rx)
        traffic_out.append(nBytes_tx)
        n = n + 1

    else: # if t_capture in [nT, (n+1)T]
        traffic_in[n-1] = traffic_in[n-1] + nBytes_rx
        traffic_out[n-1] = traffic_out[n-1] + nBytes_tx

    nData = nData + nBytes_rx
    nPacket = nPacket + 1
```

Listing 2: Traffic curves setup

After the for loop the program will have all the lists ready to print the retrieved data. First of all we print the total capture duration, the total bytes processed and the total packets exchanged. Than for each MAC address registered during the processing time we print the vendor to which it belongs to, retrieved using the IEEE standard file, and all the data retrieved during the capture. Those data are also plotted in three graphs using matplotlib. The first two graphs shows some histograms about received and transmitted bytes and packets for all the registered MAC addresses. In order to plot only relevant information we decided to set a variable called `plot_ratio` used to plot only MAC addresses which have transmitted or received at least plot_ratio% packets compared to the MAC address which sent or received the maximum amount in the capture.
The last graph will show the traffic curve of the the received and the transmitted packets.

3 Examples

To test our program we did some captures in different moments of the day to see what we are able to understand from it.

3.1 Sonos speakers interconnected via WiFi

A capture that brought something interesting to our attention is the `Filtered_capture_SONOS_WEDNESDAY_MILAN.pcapng` done in Milan on a wednesday morning. Thanks to the vendor classification we spotted a four MAC addresses related to **Sonos** devices, a vendor famous for wireless home sound systems in which multiple speakers are interconnected using WiFi. In this capture we found four different devices which produced most of the traffic, the MAC addresses of the Sonos devices are:

- 00:0e:58:c2:48:5f
- 00:0e:58:c2:48:05
- 00:0e:58:67:a3:e9
- 00:0e:58:f4:7a:83

As we can see in Figure 1, there are a lot of traffic spikes mainly due to the communication between those speakers. The highest peak is the combination of the communication between the Sonos devices and other communication flows.

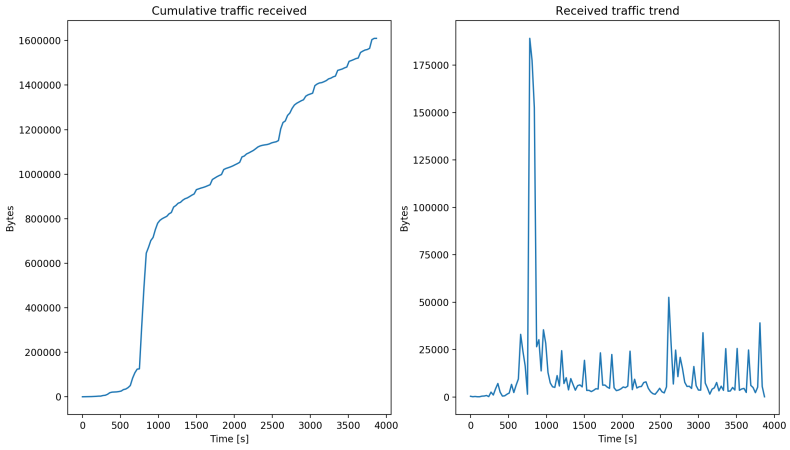

Figure 1: Sonos input traffic graphs

To further show how many packets the Sonos devices exchanged we can see that three out of the four devices are at the top spots of the following graph (2).

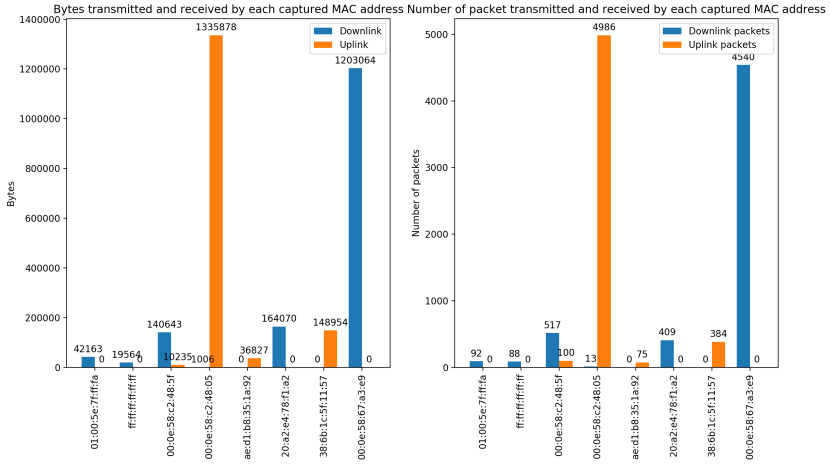

Figure 2: Sonos packets exchanged

3.2 Multimedia internet capture

In this paragraph, we analyze the results obtained with a capture performed during a lecture of multimedia internet.
The lecture has been followed using *Microsoft Teams*: the professor was presenting the topic sharing its screen and all the students were following him with camera and microphone inactive.
In the Figure 3, we can see on the left the graphs reporting the number of bytes transmitted and received by some of the *MACs* we have revealed and on the right the number of transmitted and received by the same *MACs*.
As we can see, there are some *MACs* that are exchanging much more traffic than others:

- **88:ae:07:3d:8a:30**: this is the device (an *iPad Pro*) from which the lecture was being attended. Even if the capture lasted only 5 minutes, we can see that this device has received a lot of bytes (about 42.660 MB): this is of course due to the fact that the device was using a video-conferencing application.
  If we consider the number of packets received by this *MAC*, we can compute the average length of the packets during the 5 minutes of the capture:

$$Average\ packet\ length = Number\ of\ bytes / Number\ of\ packets = 594 Bytes \tag{1}$$

- **20:b0:01:22:22:66**: this is the access point of the home were the lecture was being attended. Indeed, we can see that the access point has sent

a lot of bytes and packets: most of them were probably destined to the *iPad Pro* that was being used to follow the lecture, since that device was connected to this access point while attending the class.

Among the other *MAC* addresses present in the graph but that have exchanged only a few packets in compared with the 2 ones mentioned above, we recognize only **dc:a9:04:91:42:b9**: it's a *MacBook Pro* in the same home. During the 5 minutes of the capture, this device was being used for smart-working purposes.s

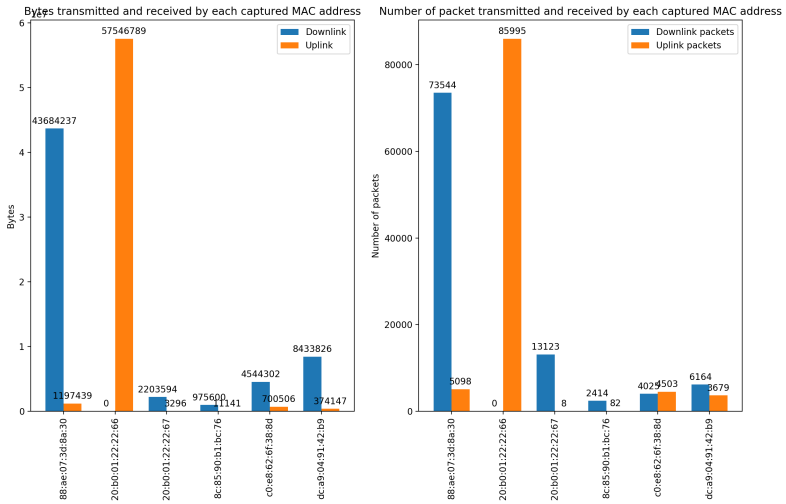

Figure 3: Multimedia internet lecture: data and packets exchanged.

3.3 Zoom capture

In this capture we were able to capture a really big flow of data due to a Zoom call activated from one of our known devices, as we can see in Figure 4 there are two devices which creates most of the traffic are:

- **8c:85:90:b1:bc:76**: A MacBook from which the call is activated
- **20:b0:01:22:22:66**: The router to which the MacBook is connected

Since all the devices involved were known, we were able to pair the result of the analysis with our knowledge on what was really being done at the moment to assign a real meaning to the packets flow captured. In this case most of the packets on the client side come from downlink since the client has to receive data on video and audio of all the other participants to the call. On the other hand the router is sending all those messages to the client so it will register an

intense uplink traffic. We can also see that the number of uplink packets in the router is bigger than the number of downlink packets sent by the MacBook since also other devices were connected to the same network at the same time.

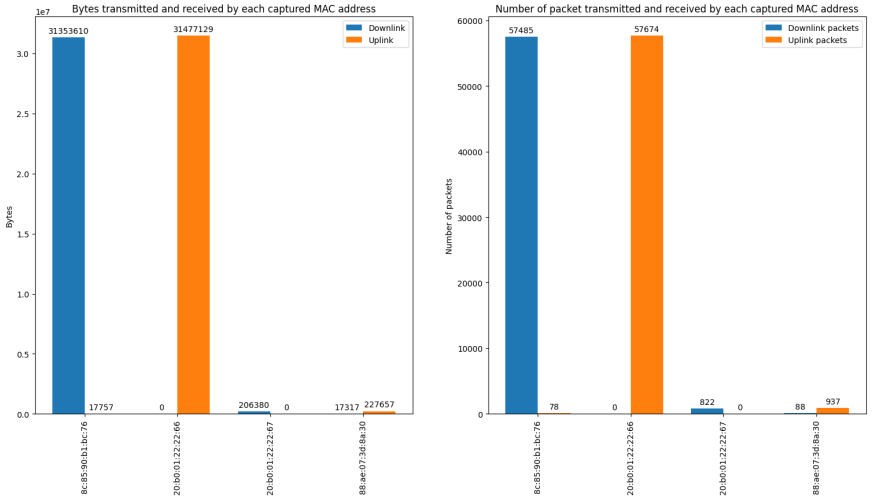

Figure 4: Zoom packets exchanged

3.4 Launch time capture