Wireless internet - WiFi traffic classification.

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

A.Y. 2019/2020

Contents

1 Introduction. **2**
1.1 Our analysis. 2

2 Project description. **3**
2.1 Program setup 3
2.2 Program structure 3

3 Presentation of results. **5**
3.1 Sonos speakers interconnected via WiFi. 5
3.2 Multimedia internet capture. 6
3.3 Zoom capture. 7
3.4 Sunday launch time capture. 8

4 Conclusions. **11**

List of Figures

1 Sonos input traffic graphs 5
2 Sonos packets exchanged 6
3 Multimedia internet lecture: data and packets exchanged. 7
4 Zoom packets exchanged 8
5 Sunday launch packets exchanged. 9
6 Sunday launch input traffic. 10

Listings

1 Packet filter 3
2 Traffic curves setup 4

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 for our project in 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 limitation 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 met:

- 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: 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 of the program looks 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:

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

Listing 1: Packet filter

Thi filter selects 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, the program will extract the destination address and add it to a dictionary which will store the number of bytes in the packet, extracted from its `packet.data.len` field, 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 number of bytes processed and the total number of packets exchanged. Than for each MAC address registered during the processing of the capture, 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 show some histograms about received and transmitted bytes and packets for some of the registered MAC addresses; we decided not to insert all of them because we wanted to plot only relevant information. To do this, we used a variable called `plot_ratio`: only the data relavite to the *MAC* addresses that have transmitted or received at least plot_ratio% packets compared to the *MAC* address which sent or received the maximum amount in the capture are plotted in the histograms.
The last graphs show the traffic curve of the the received and the transmitted packets.

3 Presentation of results.

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

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 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.

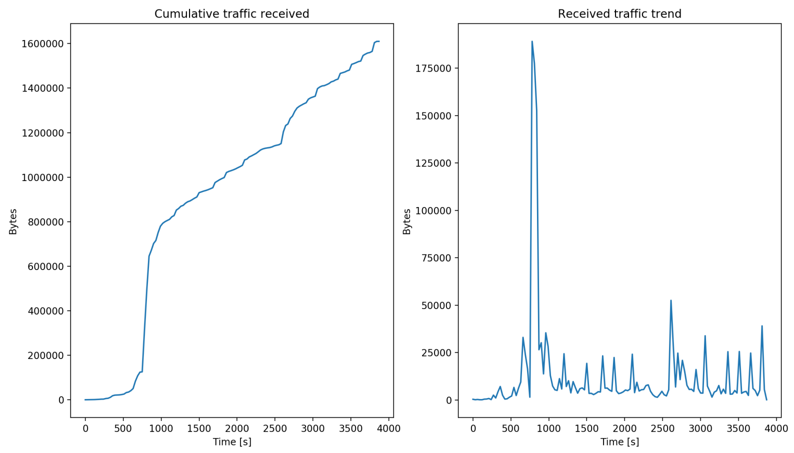

Figure 1: Sonos input traffic graphs

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 (Figure 2).

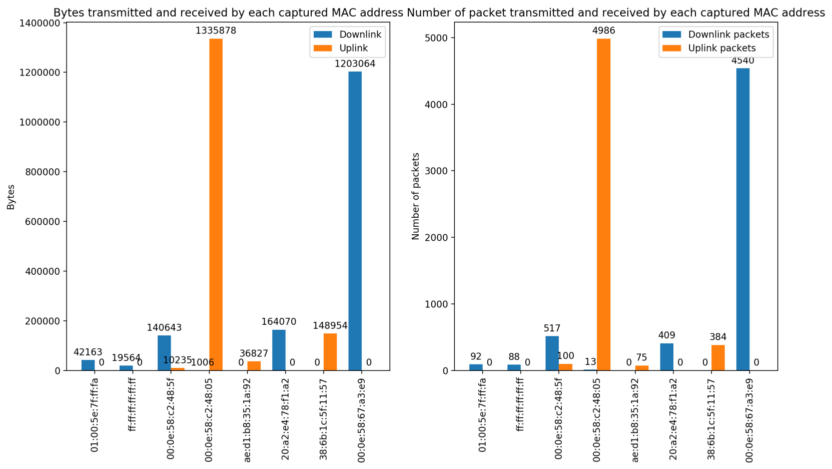

Figure 2: Sonos packets exchanged

3.2 Multimedia internet capture.

In this paragraph, we analyze the results obtained with a capture performed during a lecture of multimedia internet (`Multimedia_internet.pcapng`). 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 their camera and microphone turned off.
In 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 receiving data from 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.

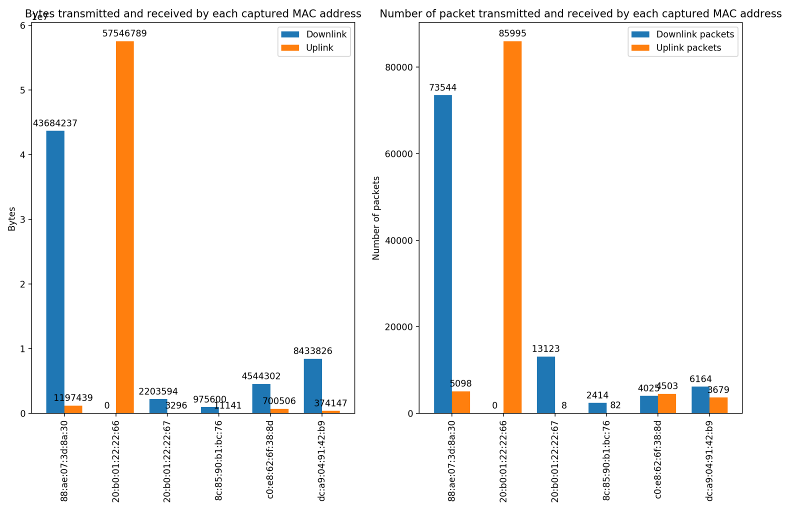

Figure 3: Multimedia internet lecture: data and packets exchanged.

3.3 Zoom capture.

In this capture, we were able to see a really big flow of data due to a *Zoom* video 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**: the 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 happening 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.

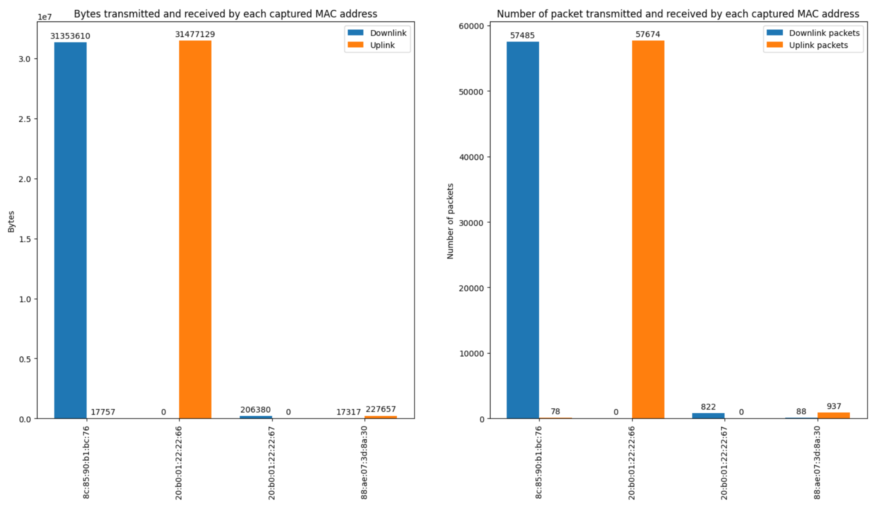

Figure 4: Zoom packets exchanged

3.4 Sunday launch time capture.

In this paragraph, we present the results coming from a capture that has been done in Livorno on a Sunday, in particular at launch time, from around 1.00 p.m. to 2.35 p.m. (`Sunday_launch.pcapng`)
In Figure 5, we can see the amount of bytes (on the left) and packets (on the right) transmitted and received by the revealed *MACs*. The ones we are able to recognize are:

- **20:b0:01:22:22:66**: the access point of the home where the capture has been performed.
- **88:ae:07:3d:8a:30**: an *iPad Pro*.
- **8c:85:90:b1:bc:76**: a *MacBook Pro*.
- **c0:e8:62:6f:38:8d**: an *iPhone*.
- **d0:c5:f3:ab:82:63**: another *iPhone*.

- **c0:9a:d0:e4:35:a5**: a third *iPhone*.

All of the just mentioned devices belong to users living in the home where we have done the capture, and are the ones the have exchanged most of the traffic; the other devices are unknown to us.
As we can see, the device that has received the highest number of bytes/packets is the access point (**20:b0:01:22:22:66**), which also has sent no bytes to any device.
Looking at Figure 5, we can also see that 3 of the known devices (**88:ae:07:3d:8a:30**, **d0:c5:f3:ab:82:63**, **c0:9a:d0:e4:35:a5**) have sent quite a lot of traffic.
Moreover, if we look at Figure 6, we can see that the amount of traffic exchanges is very low for the first (about) 4800 seconds (80 minutes) and that it grows a lot in the last 15 minutes of the capture, so almost all the traffic has been sent to the access point mainly from the 3 just mentioned devices in those last 15 minutes.
Therefore, what we can conclude is that the owners of the devices haven't used them while having launch and have started sending data to the network again once they have finished to eat, which is something very reasonable.
For what concerns the other devices, the same reasoning applies to them too, with the difference that they have exchanged only a few traffic if compared with the one of the 3 devices mentioned above.

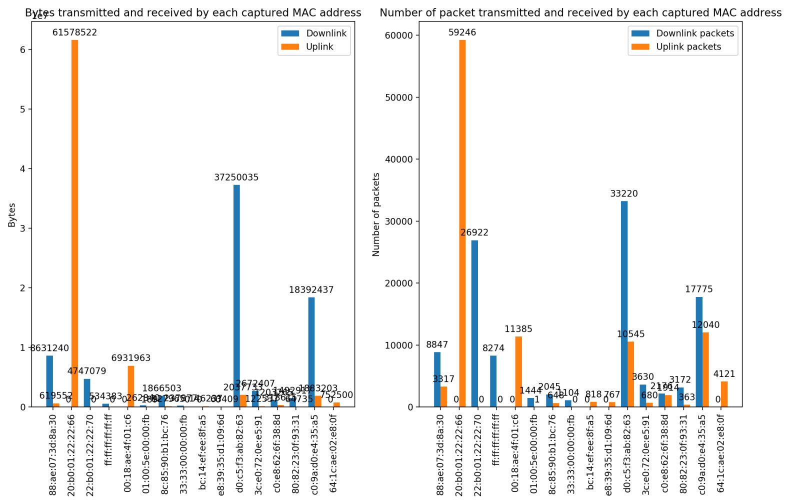

Figure 5: Sunday launch packets exchanged.

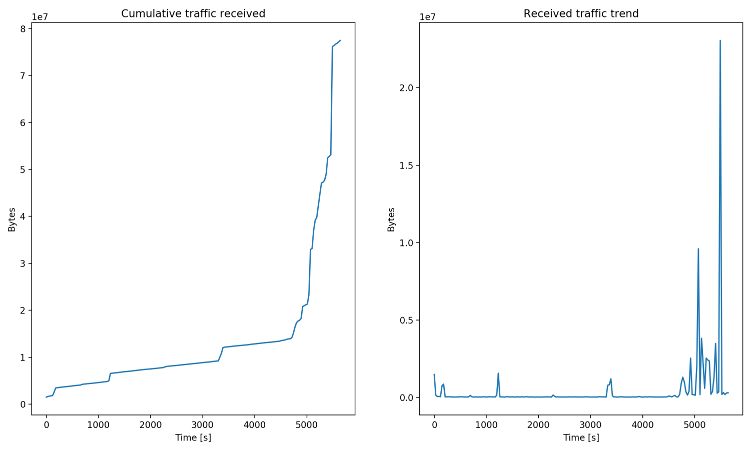

Figure 6: Sunday launch input traffic.

4 Conclusions.

Summing up, in this project we have analyzed some probe captures in order to try to classify traffic at *MAC* layer with all the limitations of this way of working.
The main results we have been able to obtain are:

- Recognition of 4 Sonos devices communicating one with the others via WiFi.
- Recognition of 2 multimedia flows, one during a zoom video-conference, the other during a university lecture, with the individuation of the devices acting as clients and as servers in the two cases.
- Recognition of a situation in which devices are not supposed to be used a lot (launch time) as expected.