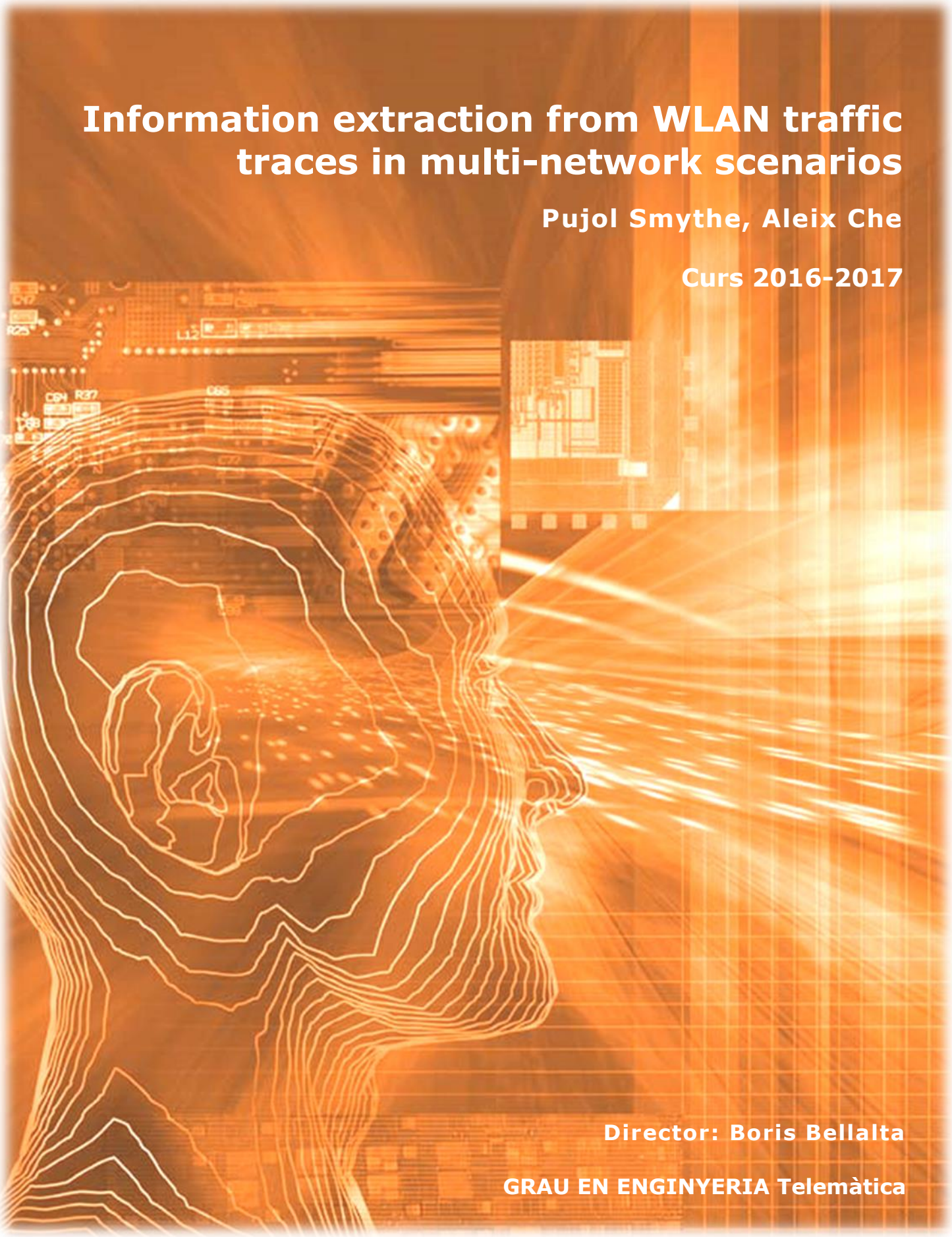
Information extraction from WLAN traffic traces in multi-network scenarios

Pujol Smythe, Aleix Che

Curs 2016-2017



Director: Boris Bellalta

GRAU EN ENGINYERIA Telemàtica

upf. Universitat Pompeu Fabra Barcelona
Escola Superior Politècnica

Treball de Fi de Grau

Information extraction from WLAN traffic traces in multi-network scenarios

Aleix Che Pujol Smythe

TREBALL FI DE GRAU

Telematics Engineering

ESCOLA SUPERIOR POLITÈCNICA UPF

2017

DIRECTOR DEL TREBALL

Boris Bellalta

Special Thanks

I would like to thank my project tutor, Boris Bellalta, for helping and orienting me during these months every time I needed as well as his advices during the development of the project. Also thank Luis Sanabria for trying and solve the problems I had with Alix even though we ended up not using them. To all the people that have participated with concluding my University Stage.

Abstract

The aim of this project is to examine how the performance of a wireless network in a zone is being influenced by its neighbours and, if so, modify their configurations to improve its efficiency. To be able to do it, we are going to analyse in a passive mode the traffic from a set of WLANs to extract relevant information to know their functionalities, and the efficiency of each Access Point detected. Once obtained, we use a program to organize all the data and obtain the correlation between each pair of devices to decide if the activity of one of them is influencing the other one. In addition, this tool will give us other important information with respect to the network. With all these results you can establish changes in the Access Points configurations so they operate in their maximum efficiency.

Contents

1. Introduction

1.1. Motivation

Nowadays, we live in a world submerged in technologies where most of the people have a big dependency. One of the most important ones is the appearance of Internet. The access to it, started only by wired connections and with low transmissions rates but it has constantly evolved into what we have now. Currently, the new technologies have improved these connections by having it wireless. The advantage of this is that any user can move freely without wires and still be connected to the network. This is possible thanks to WiFi, who uses the Standard 802.11 to make these connections.

However, to be able to carry out these transmissions, each access point needs to use certain limited resources from the channels available in the 2,4 GHz and 5 GHz frequency bands. Sharing the limited resources between multiple wireless networks can lead to congest the channel and make networks to slow their transmission rates. This congestion is due to the amount of data that multiple networks want to send at the same time. To avoid packet losses in these networks there is the CSMA/CA protocol, created for wireless and wired networks, where all devices in a same channel try to send data at the same time. CSMA helps to mitigate the collisions by providing fair access to the medium in each transmission. When reservation is done all the other devices have to wait until it finishes. This gives us the possibility to know whether a network is influencing his neighbour's traffic by looking the correlations.

To check whether these correlations reflect how the networks are affected by their neighbours, this project will present an analysis method to confirm.

1.2. Objectives

The main objective of this project is to develop a wireless network analyser which tells you the possible correlations between a network and its neighbours and how it affects their metrics like the throughput or packet rates.

To do so, we want to create different lab scenarios to test whether the analyser software works. Once checked we have satisfactory results, we want to apply the tool to real environments to corroborate it works and get to the conclusions if a channel is oversaturated or not.

1.3. Document Structure

This project will be divided in 6 sections. The first section (Introduction), will explain an overview of the project. We then have the second section (Background), where it is going to be explained how WiFi technology works and the most relevant protocols. Section 3 (Capture traces) will be in charge of explaining how to configure the devices to start capturing wireless traffic and what information from the traces obtained is relevant. Later, in section 4 (Experimental Validation in the Lab) we will specify the necessary equipment to make our lab scenarios, together with the results obtained. The penultimate section 5 (Experimental Validation in Real Environments) will be in charge of capturing traffic from real life and analysing it to check for congestion problems in the medium. Finally, we have section 6 (Conclusion) to indicate the conclusions of the project and what would be the next steps to continue the presented work.

2. Background

Our objective in this project is to, without knowing anything about the networks around us, create a tool which analyses in a passive mode the traces in a zone to observe the efficiency of them. To achieve this, we will sniff the wireless traffic to get all the 802.11 packets. We will take into account information such as the efficiency of each network, the number of Access Points (APs) around the sniffer, the correlation between networks, etc.

2.1. Wi-Fi

Wi-Fi is a Technology that appeared at the very end of the XX century and which uses Wireless Local Area Networks (WLAN). This method is an alternative to the common Local Area Networks (LAN) where the stations have to be directly connected through a wire. In addition, WLAN has the advantage of supporting moving devices, such as mobile phones or laptops, which receive connection in a wireless mode, so that the user is able to move freely through a limited zone. All this is due to the connection of the devices through radio frequency links.
The radio frequency links transmit and receive data through electromagnetic waves, which are usually in the frequency band of 2,4 GHz, although they can also work in the 5 GHz band. All of this is based in the 802.11 standard, invented in 1999. [1]
The Technology leads to the user having a greater mobility since it minimizes all the possible wires in the coverage area and therefore it gives you more freedom. Nowadays, the utilization of this type of networks has been increasing to the point where most of the devices we find support this kind of technology.

2.1.1. Topology

In a WLAN network, we can find 3 types of topology:

a) Basic Service Set (BSS)

In this mode, you connect a LAN network to a WLAN network using a device called Access Point (AP) which acts as an intermediary between both networks. These APs are in charge of the transmission and reception of the data to the end devices, also called stations (STA). In this kind of topology, the STAs cannot communicate directly between them. All the communications must go through, firstly, the AP so it can inspect and manage the information received. In each BSS we can find a unique basic service set identifier (BSSID) which usually is the MAC address of the AP to be able to identify it.[2] [3]

b) Independent Basic Service (IBSS) or Ad-hoc

In this mode, the wireless communication between STAs is done directly without the need of the APs. The STAs are connected in an arbitrary and dynamic form where they all work as routers. The coverage of the Ad-hoc is limited with the scope that has each STA. [2] [3]

c) Extended Service Set (ESS)

In this mode there is a group of BSS infrastructures connected between them using a distributed system. To be able to get this topology, with the service known as roaming, the different APs from each BSS communicate between each other to allow the different STAs to go through a BSS to another without losing the connection. [2] [3]

2.1.2. Standard 802.11

This Standard has been defined by the Institute of Electrical and Electronics Engineers (IEEE) to be able to implement WLAN networks. At the very beginning, it provided data rates up to 2 Mbps.[4] However, these rates have been improving over the last few years. The reasons for this improvement are the variants that have appeared during these years, which attempt to optimize the Bandwidth or specify components in a better way, so they can obtain a higher security. The most highlighted variants that we are going to see in our captures are:

a) 802.11b

In this case, it uses the 2,4 GHz frequency band with a maximum data rate of 11 Mbps. When this standard appeared, it ended up being the principal WLAN technology. The advantage of this variant is its compatibility due to the low cost compared to other standards. On the other hand, the maximum data rate of 11 Mbps is a disadvantage. In addition, the probability of having interferences is higher because of the bands it uses. [5]

b) 802.11g

This variant is in charge of combining the advantages of the standards mentioned above and 802.11a. It allows the networks to be in the 2,4 GHz band having Bandwidths like the 802.11a standard with the compatibility 802.11b standard has. [5]

c) 802.11n

This variant is compatible with all the standards explained before and works in 2,4 GHz or 5 GHz frequency bands. In addition, the maximum data rate, compared to the other ones, increases considerably (up to 300 Mbps). [5]

2.1.3. Channels and frequencies

As stated before, the standards 802.11b, 802.11g and 802.11n use the 2,4 GHz band, where you can find 14 channels. In Europe, you use the first 13 channels with 5 MHz width. These channels are not totally independent between themselves because the adjacent ones are overlapping, which might cause interference. Therefore, it is recommended to only use 3 channels simultaneously. The ideal non-overlapping channels are 1, 6 and 11. In this figure, we can see how the channels work and overlap. [6]

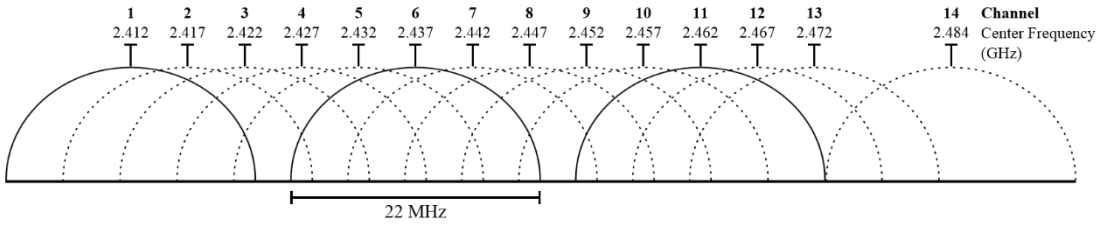


Figure 1: List of 2,4GHz channels [1]

We then have the standard 802.11n which can use the 5 GHz band. In this band, there are 23 useful channels with 20 MHz for each one. Like the 2,4 GHz, these channels are not completely independent due to overlapping. In this case, it is recommended to use 12 non-overlapping channels simultaneously. [6]

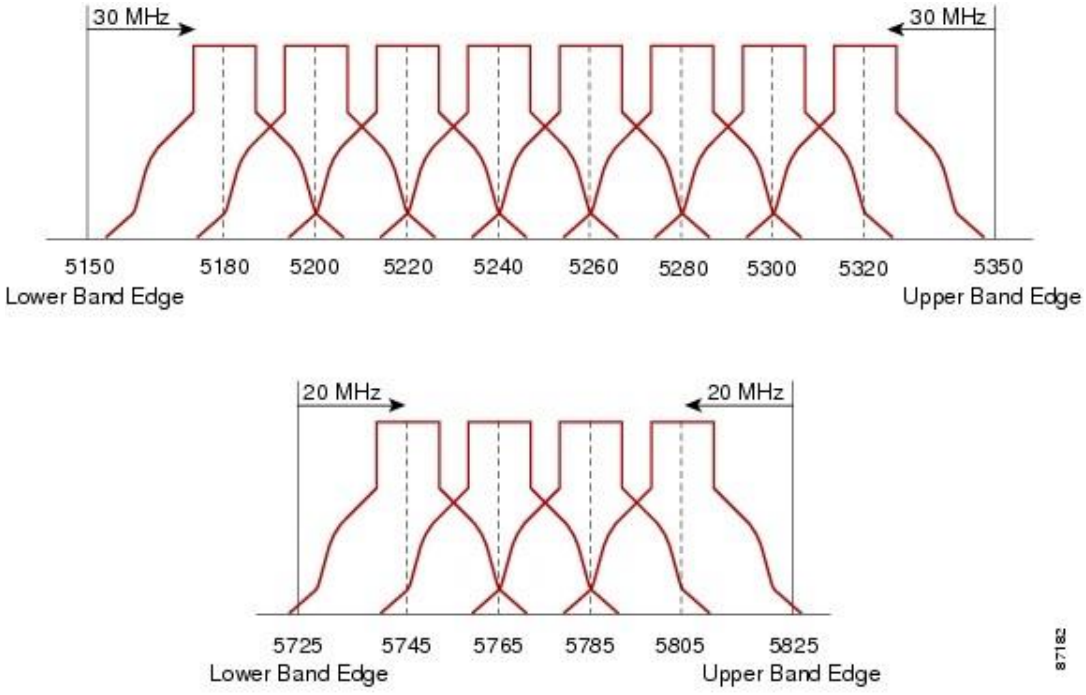


Figure 2: List of 5GHz channel [2]

[1] Retrieved February 25, 2017 from https://www.homenethowto.com/wp-content/uploads/2_4-GHz-Wi-Fi-channels-802_11bg-WLAN.png

[2] Retrieved February 25, 2017 from http://www.networkcomputing.com/sites/default/files/resources/nwc/channel%20inteference-Cisco.jpg

The utilization of more than 3 channels in one case (2,4 GHz band), or more than 12 channels in the other case (5 GHz band), does not mean it is not possible. They will find more interferences and this will affect the throughput of the WLAN with high traffic. However, it can still be done. The utilization of more channels will make packets appear from other channels in the current channel due to overlapping.

2.1.4. Frame Types

There are only 3 different types of 802.11 frames that we can find, but subtypes also exist. These types exist to enable the management and control of wireless links.

a) Management Frames

The management frames are used mainly for the association process. This means that stations can join or leave a Basic Service Set thanks to these frames. To sum up, they are in charge of maintaining and establishing communication to support the authentication, association and synchronizations.
In these frames, we can find subtypes such as Authentication, Deauthentication, Association Request and Response, Probe Request and Response, Reassociation, Disassociation and beacon frames. [7]

b) Control Frames

The control frames are used to collaborate on the delivering of the data frames between STAs. These are usually transmitted in a basic rate because they need to be listened by all the stations. The frames try to avoid collisions and clear the channel of transmissions.
The subtypes we can find in these kinds of frames are Request To Send, Acknowledgement and Clear To Send frames, among others. [7]

c) Data Frames

The data frames are in charge of transporting the information from upper layers. They can only be send after the communication between AP and STA is established. [7]

2.1.5. CSMA/CA

In an environment with a shared medium, like the Wi-Fi has, all devices have a granted access to it with no concrete priority. Therefore, if more than one host creates a transmission simultaneously, signals will collide and the network will have to recover so it is able to continue the communication.

To solve this problem, the *Carrier Sense Multiple Access with Collision Avoidance protocol* (CSMA/CA) will be on charge of the access to the network by allowing multiple stations to use the same transmission medium. To do so, before they send the message, each device announces his intention of transmitting to avoid any collision. This will allow hosts to know when a collision has occurred and, instead of sending the trace when the medium is free, will wait an additional random time to retransmit his intention of transmitting. [7]

If a sender station wants to send a message, he first sends a RTS (Request to Send) trace. If the end device receives the trace, it means he is prepared to receive the message. Therefore, it will send a CTS (Clear To Send) packet to the sender so he starts to send the message. In case there is a collision and the sender does not receive any CTS, after the waiting time, he is going to wait an additional random time before retransmitting the RTS until it receives a CTS as a response. Finally, when all the message are sent correctly, the receiver will send an ACK to the sender to close the communication.

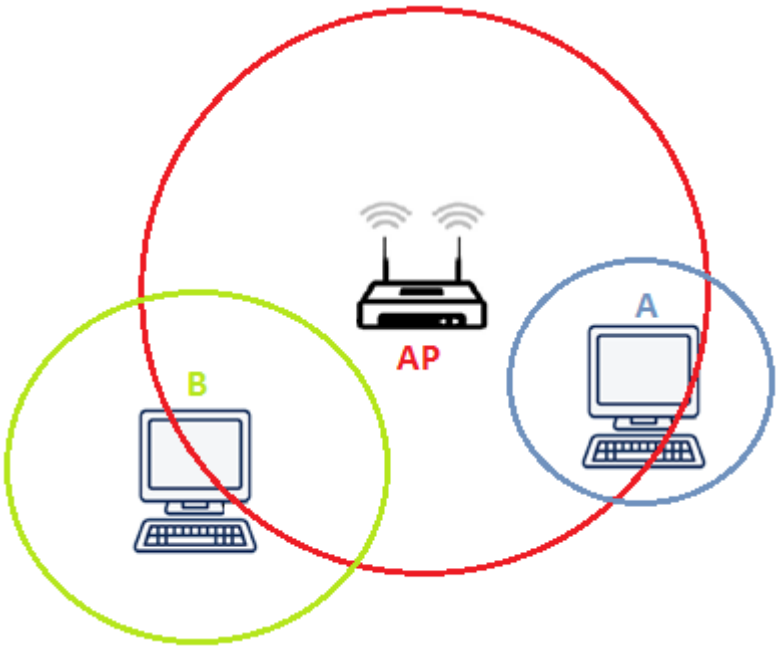


Figure 3: Distribution of a Wireless Network

If we consider this example, the AP can communicate with A and B but both computers do not know they exist and cannot communicate directly. This is a problem because A could generate a collision in the channel if B sends packets at the same time, as seen in the next figure.

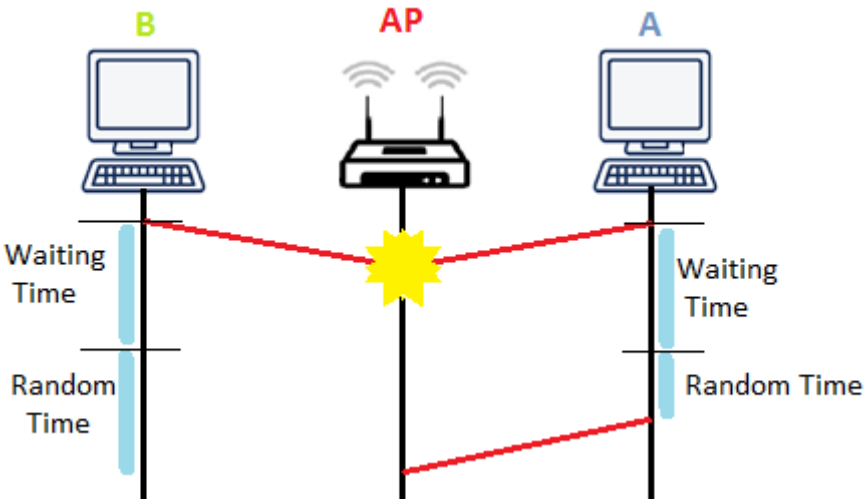


Figure 4: Packet collision when sending RTS

Once each computer sends the RTS, they both have a waiting time to receive a CTS. In case they do not receive it, they wait a random time so the collision does not appear again. After all this time, A sends a new RTS message before B because he had a smaller random time.

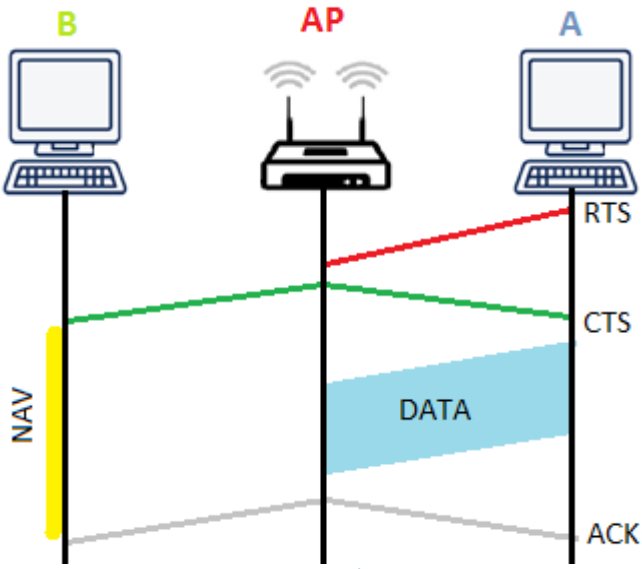


Figure 5: CSMA/CA dialog

At this time, AP has received the RTS packet, so sends a CTS to all the nodes in the area and waits for the data to come. A, once received the CTS, starts sending his data whereas B stays in standby. Finally, when all the data is send, the AP sends an ACK to acknowledge the packets have been correctly received.

3. Capture of traces

To proceed with the capture of the wireless traffic we need to know what information would be useful to analyse. Once we choose the relevant information, we will continue by configuring the sniffer enabling it to capture the 802.11 traces.

Finally, when the capture is already done, we will export the data as a csv file, which will contain the previously selected information. This will allow our program to import all of the data and, with the help of RStudio, process a high quantity of data in an efficient manner.

3.1. Data to consider

During the project, we must calculate important data to determine if the wireless network is optimal for data transmission and if the networks are correlated between them. To do so, we will first have to extract the relevant information from the capture. We will proceed all this information to obtain relevant results.
If we want to extract this information we will first analyse the tree view of the packets seen in Wireshark, decide what fields in the view are relevant and assign them into columns so it can be seen in the csv file.

3.1.1. IEEE 802.11 type field

Inside this label we can extract important information to know such things as the source and destination addresses, the BSSID or type of packet. [9]

a) Type/Subtype

In this field, we can usually find the subtype of the frame. In 802.11 packets, we can find 3 frame types previously explained with many different subtypes each one. We will need to represent this field in the columns to be able to classify the packets in each of the frame types. The subtype will be important to be able to detect the Beacons and, therefore, knowing the APs the sniffer has been detecting.

b) Type

Inside the Frame label we will be able to find in which of the 3 frame types the packet belongs. The type field will be important to classify the packets and get the number of packets each device has been sending over the total time it has been sniffing.

c) DS Status

If we go deeper into the frame label we will find another label called Flags. Inside there we will take into consideration the DS status. It tells you whether the packet is being transmitted from an AP or not, if it is AD-Hoc network or if the packet is being send to an AP. DS status will be important to be able to know the total time each device is using the channel during the sniffing time.

d) Transmitter & Receiver address

This fields will tell you where the packet comes from and where it ends. It is very important to differentiate from Source and Destination addresses because, although they can be the same in Management frames, in Data frames can be different. In the first case it tells you the addresses of the devices the packet is going through at this moment and, in the second case, it tells you the original sender and final destination. We will need this fields to get from which AP the packet is going or coming from. Therefore, we will able to get the amount of time an AP is using a channel.

e) BSS Id

To be able to know from which BSS the packet belongs, we will need to see this field, previously explained in section 2.

3.1.2. 802.11 radio information

In this label, we are able to get information about the medium. [9]

a) Channel

This field will tell you the channel in which the sniffer detected the packet. This will be relevant to know if the packet is supposed to be in the current channel.

b) Signal Strength (dBm)

This field will tell you the Signal strength of the BSS with respect to the sniffer. What this field measures is the received signal strength indicator (RSSI). The value of this field is usually between -30 and -100 dBm.

c) Duration

This field will tell you the amount of time it takes the packet to be transmitted. We will use this to know the total time an AP is using a channel.

3.1.3. IEEE 802.11 wireless LAN management frame

This frame has the parameters each device is capable of. It is only available in the Management packets. [9]

a) SSID

This field will tell you under what name the AP is working. The SSID, as said before, will be the visual name of the network so the users can be able to detect and register into a wireless network of their scope.

b) Current Channel

This field will tell you in which channel the AP is working with. It is a parameter to consider for discarding the packets that have been detected in a wrong channel.

3.2. Metrics

With all this fields explained we can obtain different metrics to consider and analyse if a channel is being oversaturated. The metrics we have considered more relevant are going to be explained in the next subsections.

3.2.1. Throughput

The Wi-Fi communications are used, as the LAN networks, to be able to transmit data between stations. To know how fast this data can be transmitted we have this metric. The throughput is the rate of the number of bits successfully send as data from the total time of the capture and it is usually measured by bits per second (bps).
If we look to our program, we can see that the function in charge of obtaining this results is number 3. What the created tool does is to obtain the total number of data bits sent in all the sniffing and divide it by the time it has been capturing. With these results, we can check the amount of bandwidth each AP is using at this moment and check if there is an AP monopolizing the channel and, therefore affecting their traffic.

3.2.2. Correlation

To be sure if two APs traffic are affecting between them we look at their correlation. This metric is a statistic measurement that indicates the force and direction of a linear relation between two random variables. In this occasion, we are going to use the Pearson correlation with the next formula. [10]

$$r = \frac{n \sum xy - (\sum x)\ \sum y}{\sqrt{\left[n \sum x^2 - (\sum x)^2\right][n \sum y^2 - (\sum y)^2]}}$$

Where *r* is the correlation coefficient, *n* the number of values in each data set, *x* the values of one AP in this case and *y* the values of the other AP.

Because of the utilization of CSMA in the wireless networks, each AP waits for all its neighbours to stop transmitting so it is able to use the channel. This waiting period would end with a negative correlation in case an access point is impeding another one to transmit until it finishes.

In our case, with the help of the tool created, we are going to perform a table for each AP that tells you the percentage of time it is using the channel every second. The table is obtained in the function 5 of our code. With this information, we can get the values of each pair of APs in each second and check if there is a correlation between them. If the values of an AP increase whereas the other decrease, we can observe a correlation that it is affecting their resources. This value can oscillate between -1 and 1. In our case we are going to considerate the values that get closer to -1 which means there is a strong negative relationship. The correlation is obtained in functions 6 and 7 of the code.

3.2.3. Retransmissions

Another important metric to check if an AP has a correct traffic flow is looking at the number of retransmissions in their traffic. To calculate this rate, you just have to do a simple division of the number of data packets been retransmitted by the total amount of data packets. The retransmission rate can give us a clue if the resources of an Access point are being drained or not and if the throughput of traffic is brushing the maximum transmission rate of the device. In the code of our tool you can observe that this metric is being calculated in function 3.

3.2.4. Packet Rates

A useful metric to check the amount of packets each AP is transmitting is the packet rate. We are able to see the number of packets per second each device is managing. The function 3 of our tool obtains the packet rate for each type of packets. This information can help us to know what AP is more active in a channel and, combined with the other information, confirm whether a device is draining the others activity.

3.3. Capturing the wireless traffic

To start sniffing the wireless traffic of the area you first need to prepare the computer that will behave as a sniffer. To do so, it is necessary to have a PC with a wireless controller, the Wireshark software and, in this case, a Kali Linux OS booted from a pen drive.

3.3.1. Monitor Mode

To be able to accomplish these objectives we will need, first of all, a wireless adapter or computer that supports monitor mode. The monitor mode allows you to monitor all traffic received from the wireless network without the need of associating with an AP or Ad-Hoc network. To use this mode in Windows you will need the Acrylic WI-FI application which is not supported by all the wireless cards and it is expensive also. As an alternative, in our case, we will use Kali Linux[3] system to be able to sniff wireless packets. The use of this OS allows you to boot your windows computer into a USB and start it with Kali Linux. Kali is the alternative if your default OS in your computer is not Ubuntu. Having Ubuntu in Virtual Machine is not the solution because the wireless card inside the VM acts as an Ethernet connection and, therefore, you will not be able to sniff wireless packets.

To start sniffing you will have to start booting the computer from the pen drive with the Kali Linux OS. Once you are in it, you will proceed to configure the wireless adapter to monitor mode. To get there, you first need to get the name of the wireless adapter by using the next command.

```
ifconfig
```

[3] Retrieved November 20, 2016 from https://www.kali.org/

As you can see in figure 6 in our case the name is “wlan0”.

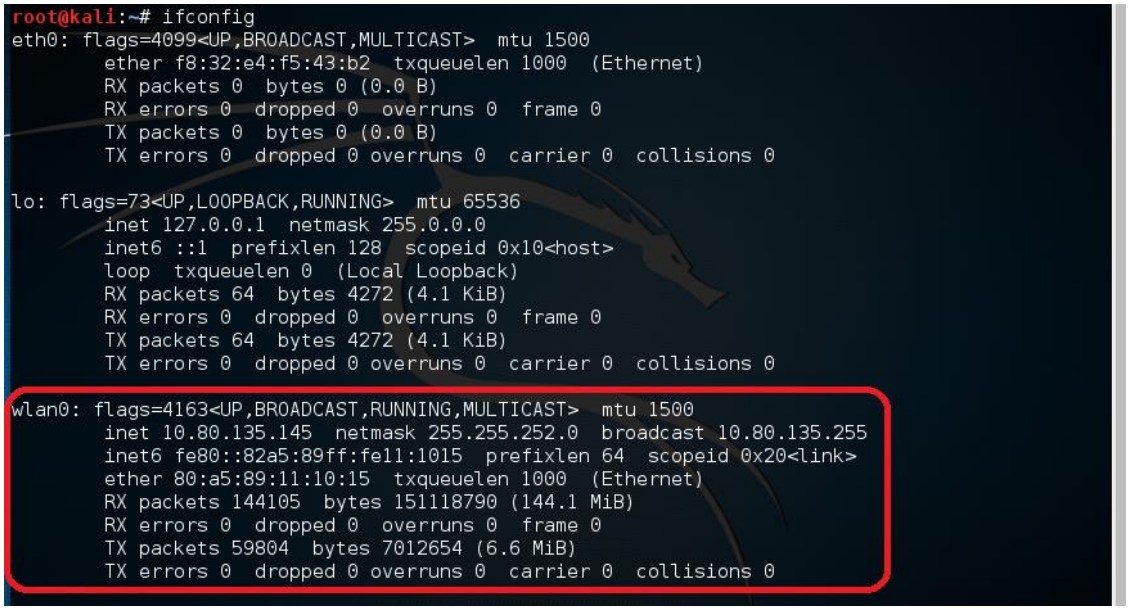


Figure 6: “ifconfig” wireless adapters information

Once we have the name, we use the next command.

```
iwconfig “name of adapter”
```

This can help you to check whether the adapter is in monitor mode or not by looking at the marked field seen in figure 7.

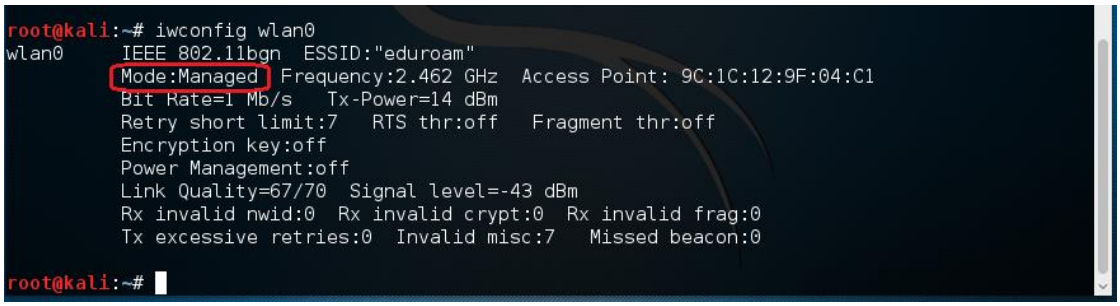


Figure 7: “iwconfig” wlan0 wireless adapter information

In case it is not in monitor mode you may need to turn off the adapter, change it to the desired mode and, finally, turn on the adapter. This is possible by using the following commands.

```
ifconfig “name of adapter” down
iwconfig “name of adaptor” mode monitor
ifconfig “name of adaptor” up
```

All this steps can be seen in figure 8.

Figure 8: Monitor mode configuration

3.3.2. Channel Hopping

The monitor mode is only able to scan a channel at a time. Although monitor mode does not support scanning multiple channels at the same time, you can create a script which hops channels automatically while you are capturing. The code will allow you to capture the activity of all the channels in the same capture.

For beeing able to do these jumps, what the script does is to repeat the command that changes the channel where the wireless adapter is listening to. The structure of the command is as it follows.

```
iwconfig "name of adapter" channel "channel you want to sniff"
```

In this project, we use a script named "changehop.sh" (**A1.Hopping Channels Script**) so it goes through all the channels by swapping between them in a desired time. The script creates an infinite loop where it uses the command shown above continuously for all the channels you want to sniff every desired time. To be able to execute this script you first have to give the corresponding permission as we see in figure 9.

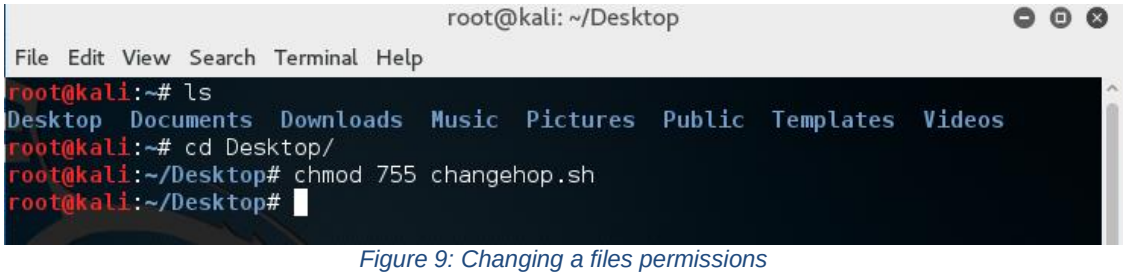


Figure 9: Changing a files permissions

Once you execute the script you can proceed to select the corresponding wireless adapter in Wireshark and start capturing all the packets.

3.3.3. Wireshark sniffing

Once we have the wireless card prepared to detect the wireless traffic and the script to hop between channels we can proceed to sniff all the traffic with the Wireshark[4] application. Wireshark is a network protocol analyser which captures all the network packets and gives as much information as possible to each of them. It is the first software which has a graphical interface to make it easier to use it. This open source packet analyser is able to capture all the network traffic information by using the pcap API. The software allows you to use network interface controllers to be able to analyse the traffic in a promiscuous mode. [8] This mode, combined with having the wireless card in monitor mode to see the 802.11 packets and using Wireshark will allow you to get a file with all the packets captured to proceed and obtain all the interesting information to analyse the traffic around the sniffer. Once we have the capture, we can extract it as .csv so it is easier to extract the useful information and discard the rest.

To start with the capture, you first need to decide whether you want to sniff through all the channels or just one. In case you want a full capture with all the channels you should run the script explained in appendix **A1.Hopping Channels Script**. If your intention is just to analyse one channel, you should use the command.

```
iwconfig "name of adapter" channel "channel you want to sniff"
```

As said before, we are going to use the Wireshark analyser. To make sure you are sniffing in the right wireless adapter you will first go to capture options as shown in figure 10.

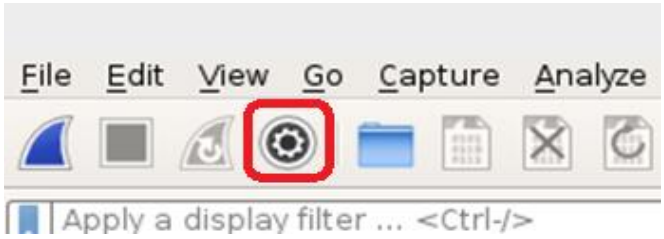


Figure 10: Capture options

[4] Retrieved December 15, 2016 from https://www.wireshark.org/

Once in there, you will then have to select the wireless card in monitor mode, change the Link-layer Header for 802.11 plus radiotap header and enable the monitor mode as shown in figure 11.

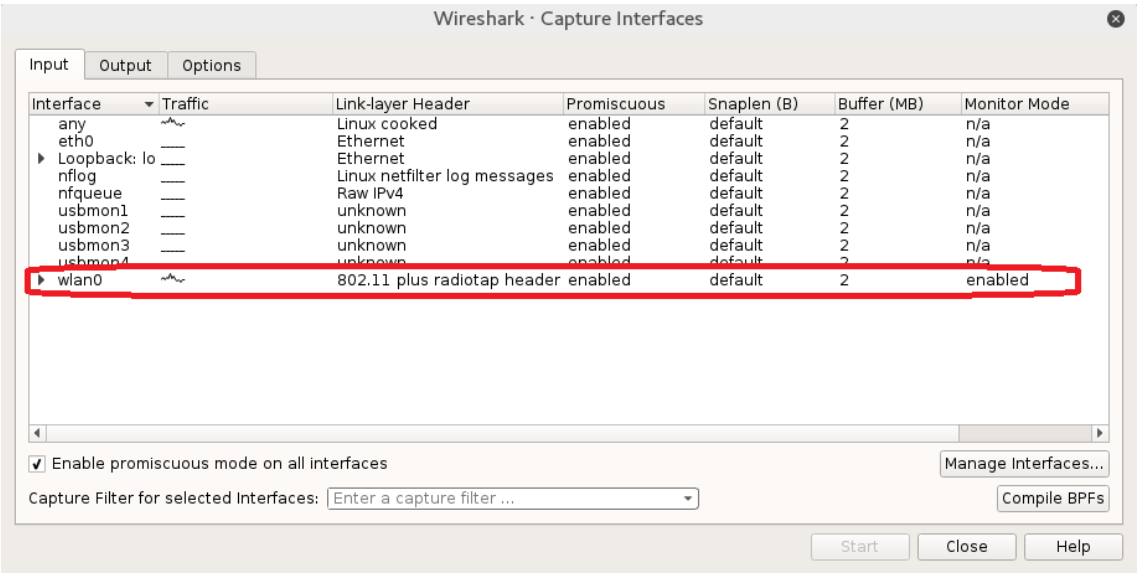


Figure 11: Capture Interfaces

Once Wireshark is properly configured, you can start the capture and run it until you have all the information you want. Finally, to be able to analyse all the traces obtained with our program you will need to extract all the information as data sets. In this case, as explained before, we will export the capture as a csv file in order to import all the traces in the R program.

To do so, you will need to go to the file option and select Export Packet Dissections as CSV as shown in figure 12.

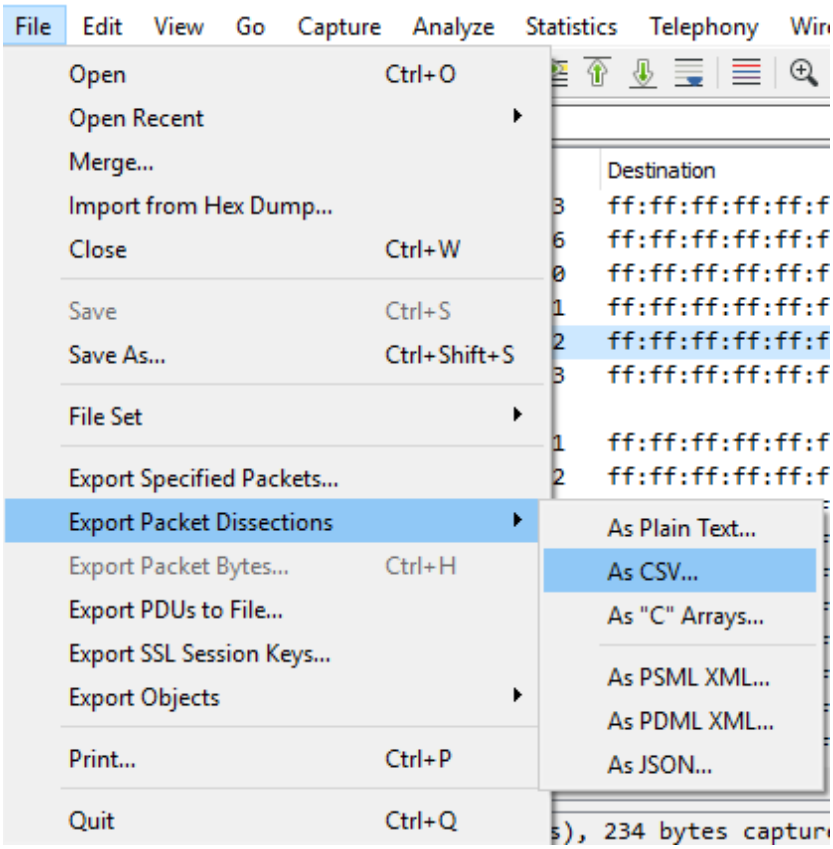


Figure 12: CSV file exportation

However, before extracting the file you will first need to select the relevant information you need to be able to analyse the traffic correctly and assigned in different columns of Wireshark. This will be explained in the next section.

3.4. Specifications of the Data Analyser Software

So that anyone can use the program created with R to obtain all the metrics that will be explained in the next sections, we posted the code with comments in github (**A3.Program code)**. You will find the code in the URL added here:

https://github.com/martiri/wifianalyser

The program created with the language named R, is an environment and programming language that focuses on statistical analysis. It is a free software implementation of S but with a statistic range support. You also have the possibility of downloading different libraries or packets with calculation or graphing functions.[11] This language gives you the opportunity of extracting and manipulating the information in an easy way. In our case, to make the coding easier we will use the Rstudio[5], an IDE (integrated development environment) for R. In this IDE there is a lot of helpful tools to make the programing faster and more organized. Before installing Rstudio you have to make sure you have installed R[6]. In our case, we will use it to process high amount of data obtained from Wireshark and get all the possible information to determine if a specific network needs improvement or not. This data will represent a high amount of 802.11 packets ordered in tables made by the program.

3.4.1. Interface of the Program

Our code will be executed, as said before, in the Rstudio IDE. This will give us the opportunity of having a easiest way to watch our data without having to create an interface from the beginning.

[5] Retrieved 15 December, 2016 from https://www.rstudio.com/
[6] Retrieved 20 October, 2016 from https://www.r-project.org/

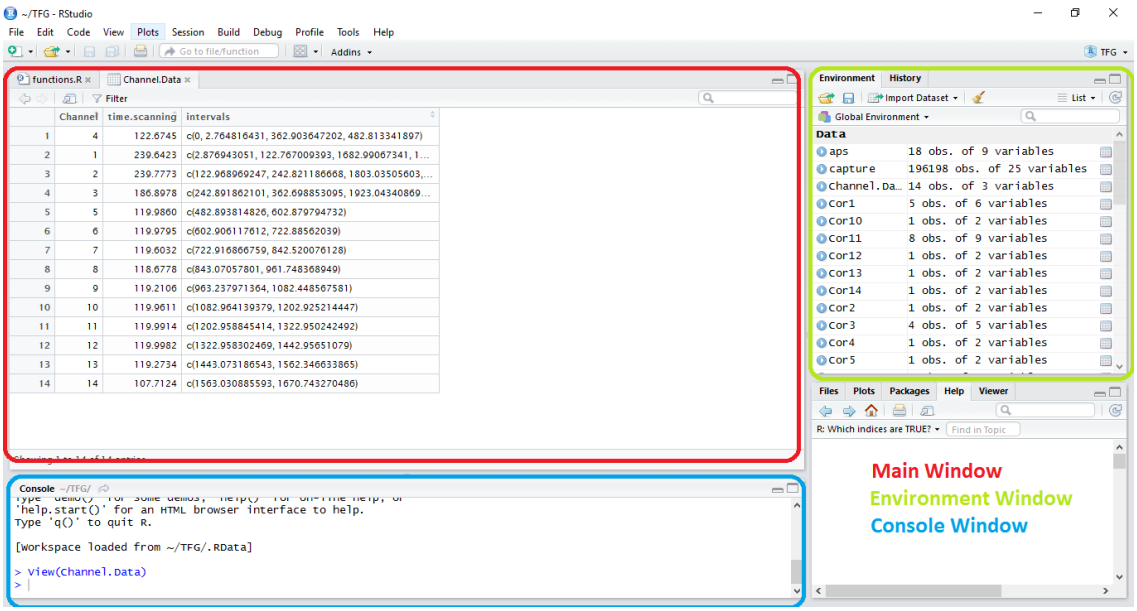


Figure 13: RStudio interface

As we can see in figure 13, there are 3 windows in the RStudio interface. In the Main Window, we will be able to modify our code and also visualize our tables created by our program. To see the tables, we need to go to the Environment Window, which has a list of all the tables created, select the one we want to visualise and the content of the table will appear in the main Window. In figure 13, as we can see, we selected the Channel.Data table to see it in the Main Window. Finally, we have the Console Window where you can write commands to modify an environment or install some additional packets.

3.4.2. Program Functionalities

Once extracted the capture as a csv file, we can proceed to collect the information obtained by using the program we have created in Rstudio. This program has in total 8 functions in charge of different tasks.

a) Export_aps_data

The first function will be on charge of importing the csv file and transform it into a data frame with the same columns as in the Wireshark capture. Not only it will organize all the data in a table, but also will create another table with all the beacons detected in the capture. In addition, after obtaining the table with all the beacons, it will create a data table named “aps” with the MAC address of all the Access Points detected, their RSSI and the SSID they belong to. To get to the last table, the function checks the “beacon” table for all the possible different BSS IDs and calculates the mean of all the packets with the same MAC address. Finally, the function will return a list named “data” containing the “capture” (all the traces), the “beacons” (beacon traces) and the “aps” tables.

b) Channel_data

The second function will create the table “Channel.Data” and will be in charge on detecting in what channels the capture has been sniffing and the seconds it has been in each one. To do so, the function needs as input the “capture” table, which contains all the traces from the capture done. In addition, it will collect the time in which it is starting and ending the analysis in each channel. This means there can be different intervals in each frequency. All of this can be done by checking the capture table and getting the columns which tell you the channel and time of each packet. The function will return the “Channel.Data” table.

c) extended_aps_table

The third function will be in charge of adding 5 columns in the “aps” table. The first 3 columns will represent the number of packets per second that have been transmitted in each AP. There are 3 columns because the program will calculate the rate of each of the 3 types of packets. To be able to get this results the program will check the type, the BSS ID, the status and the Transmitter and Receiver address fields. The BSS ID and the status will allow you to know from what AP is being transmitted or receiving the packet. The Type will allow you to classify each packet analysed and, therefore, know the amount of packets of each type there is. Then, we have the forth column, in charge of obtaining the throughput of each AP. Finally, the last column be in charge of getting the retransmission rate of each AP by looking at the Retry field.

Therefore, to be able to obtain all this metrics the function will need as input the “capture” and “Channel.Data” tables and it will return as output the “aps” table with 5 new columns.

d) sta_table

The forth function will create a new table with all the MAC addresses of the end devices, i.e. STAs, detected in the sniffer coverage. In addition, it will add 3 columns in charge of obtaining the data rates of the 3 different types of packets. This function will need as input the “capture” table

Highlight that if the medium has a lot of different STAs detected, the program can take a few minutes to obtain all the results. Therefore, if you are not interested in what the end devices are doing, it is recommended not to execute this function.

e) ap_histograms

The fifth function will be in charge of creating a histogram for each AP showing their efficiency in every second. It will plot the percentage per second the AP is using the channel. This information will be useful to detect whether an AP is draining all the resources of the channel or not. It also saves the histograms in the corresponding channel folder. To get with these functionalities the function will check the column intervals from the Channels table to get the time moment a packet was transmitted and the channel in which the packet is being transmitted. It will also check whether a packet is in the correct channel or not by looking at the BSS ID.

To accomplish this, the input of the function will be the “capture”, “Channel.Data” and “aps” tables. The output will be a “apinfo” list containing a table for each AP with the information of how much time they have been occupying the channel in each second. In addition, it will create a folder with all the histograms.

f) total_correlation

The sixth function will be in charge of looking the correlation between the Aps in a same channel to know if an Access Point is influencing in another Access Point traffic. To do so, it will need as input the “aps” and “Channel.Data” tables and the “apinfo” list. It will return a table for each channel containing the total correlation between each pair of APs.

g) correlation_xseg

This function will be a more detailed version than the *total_correlation* function. It will be in charge of checking the correlation of each pair of APs in intervals of x seconds. The interval can be specified in the global variables and must be higher than 2 seconds so it is able to work. To do so, it will need as input the “aps” and “Channel.Data” tables and the “apinfo” list. It will return a table for each channel containing the correlation calculated every x seconds between each pair of APs.

h) Relations

Finally, the last function will take into consideration the tables generated in *correlation_xseg* function to determine whether 2 APs are being influencing in their activity or not. As input it will need the “aps” and “Channel.Data” tables and the “apinfo” list. The function will return a “Relations” table for each channel containing a column with all the APs working in those frequencies and a second column containing the APs they are correlated with.

3.4.3. Metrics

Once we have clear what kind of information we want to extract from the captures, we can proceed to generate the desired metrics to determine the access points efficiency. With all this metrics, we can decide if an AP configuration needs to change to stop influencing its neighbours.

a) Access Points, Stations & Channels

First of all, we have a table named “aps” with all the Access Points detected by the sniffer. Inside it, we have metrics like the throughput, the SSID, the signal strength, the packet rate of each of the types and the retransmission rate of each device. To determine whether the AP is transmitting correctly or not, the most useful information would be the throughput, the retransmission rate and the packet rates.

Metrics	Description
BSS.ID	MAC address of each AP useful to identify them. The address is unique for each device.
SSID	The Service Set Identifier (SSID) is in charge of indicating what is the name of the Network the AP is creating.
Current.Channel	This field will indicate in what channel the AP has been working during the sniffing.
RSSI	The Received Signal Strength Indicator (RSSI) indicates the signal strength with what the sniffer has received the packets.
Management.Rate	In charge of obtaining the amount of management packets the AP manages per second (packest/s).
Data.Rate	In charge of obtaining the amount of data packets the AP manages per second (packest/s).
Control.Rate	In charge of obtaining the amount of control packets the AP manages per second (packest/s).
Throughput(Mbps)	Calculates the how long it takes to send the data bits per second (Mbps) from one point to the other
Retransmision	Rate to know the percentage of packets that have been needed to retransmit in each AP.

Table 1: Parameters in "aps" table

We also have a table named “sta” which contains all the stations detected in the area. In this table, we can observe which of them have more activity be giving us information of the packet rate for each of the 3 types.

Metrics	Description
STA	MAC address of each STA useful to identify them. The address is unique for each device.
Management.Rate	In charge of obtaining the amount of management packets the STA manages per second (packest/s).
Data.Rate	In charge of obtaining the amount of data packets the AP manages per second (packest/s).
Control.Rate	In charge of obtaining the amount of control packets the AP manages per second (packest/s).

Table 2: Parameters in "sta" table

Finally, we have the table named “Channel.Data” which give us information about the channels captured. It tells us the channel it has been sniffing and the total time it has been capturing packets on it. It also tells in what moment of the time in the capture it has been sniffing the channel.

Metrics	Description
Channel	Identifier of the channel.
Time.scanning	The total time the sniffer has been capturing in the corresponding channel.
intervals	The time in the capture the sniffer has started and stopped sniffing the corresponding channel. Since the capture can go back to the same channel in other moment there could be more than one interval in this field

Table 3: Parameters in "Channel.Data" table

b) Correlation

To know the total correlation between each pair of APs, our R program makes a table relating them. In this data frame, we can observe a value between -1 and 1 telling us how related are the APs.

We also have a table, named “segCor**X**” (where **X** is the channel from it has obtained the traces), that gives a more detailed information of each relation. In this table, you choose how many seconds you want to have the correlation of each pair of APs. In these data frames is where you can really see if there is a relation.

c) Relationships

The final table is called “Relations” followed with the channel it is analysing. Inside this table we have a list of all the APs telling you if there is a relation with others. This is thanks to the correlation obtained every x seconds i.e. the *segCor* table.

d) Plots

We finally have the plots where you can see in a more visual manner if there is a correlation. In this part, you can first choose whether you want to see the plots by histograms or by a line in time. The program will create a plot for each AP where you can see the rate of utilization in every second. This is useful to compare 2 plots and confirm there is a correlation between them.

We also have a RSSI histogram where it shows you the signal strength of each AP and where you can conclude if you can trust an AP because of its distance.

4. Experimental Validation in the Lab

When the sniffer is successfully configured, we have Wireshark running and all the code for the analysing tool is created, we can proceed to create different controlled scenarios to make sure if our objectives can be done correctly. In this section, we are going to explain all the equipment used for these experiments and how to configure them to be able to obtain the best results for our project, as well as a validation of all the metrics and results we have got from our tool.

4.1. Equipment

To be able to determine whether we can get relevant information or not with the code created, we will first create controlled cases where we can confirm it is possible to get any relevant correlations between APs. If the information extracted gives us useful information we will be able to detect if an AP is draining the resources of another one. To make these scenarios we will need the following equipment:

a) 2 Lynksys wireless routers

To accomplish our purposes, we will need to create two independent networks in the same channel with 2 different routers[7]. These routers will act as access points and will be on charge of routing the messages from the sender to the receiver. In this case, we will create two independent BSS topologies with SSID AP1 and AP2 respectively.

Figure 13: Lynksis routers

To have the desired configuration for each router we will first have to access to the router options inserting its IP address in a browser. Once inside, we need to go inside the section Wireless to Advanced Wireless Settings. Once there, we need to change the Basic rate from default to 1-2 Mbps and the Transmission rate from Auto to 1Mbps. The configuration has to be like this so it is easier to congest their mediums and detect correlations. With this options, we will not have to create big amounts of traffic to drain the resources of the APs.

[7] Retrieved May 3, 2017 from http://www.linksys.com/es/en/p/P-WRT54GL/;jsessionid=38170A5E79EEFB35F8118CE55A9795EF/

b) Iperf

To generate different scenarios for the project we will need a tool called Iperf that generates TCP and UDP traffic. This tool will simulate a communication between two hosts in the same network so we are able to sniff and, finally, analyse the traces captured. It is also used to help you and measure parameters such as the bandwidth or the quality of a network link by sending packets from a sender to a receiver. To be able to use it you will need a server, in charge of listening and receiving the packets sent from another host, and the client, in charge of sending the packets with parameters specified to the sender. In our case, we have used Iperf3, a new implementation from scratch with the goal of a simpler code base, which has additional features and did not present any problems with the devices used.

c) 2 Tablets

To simulate an exchange of data we will need 4 devices, 2 for each network. Because of the lack of computers, we used 2 Tablets to pretend they were the servers for each SSID. During this validation lab, we used the tablets Alcatel Pixi 7[8].

Figure 14: Alcatel Pixi 7 tablets

In fact, you can use any device capable of using Iperf. For the use of this tool in the tablets we downloaded an application called *Network Tools*[9]. In addition, the app will give you useful information like the IP address of the device or its MAC.

[8] Retrieved May 10, 2017 from http://www.alcatel-mobile.com/gb/pixi-3-7
[9] Retrieved May 10, 2017 from https://play.google.com/store/apps/details?id=net.he.networktools&hl=es_419

To use it correctly you must select Iperf3 in the menu option and type the following command:

```
iperf3 -s
```

Where *-s* means, the **device** is listening for future communications.

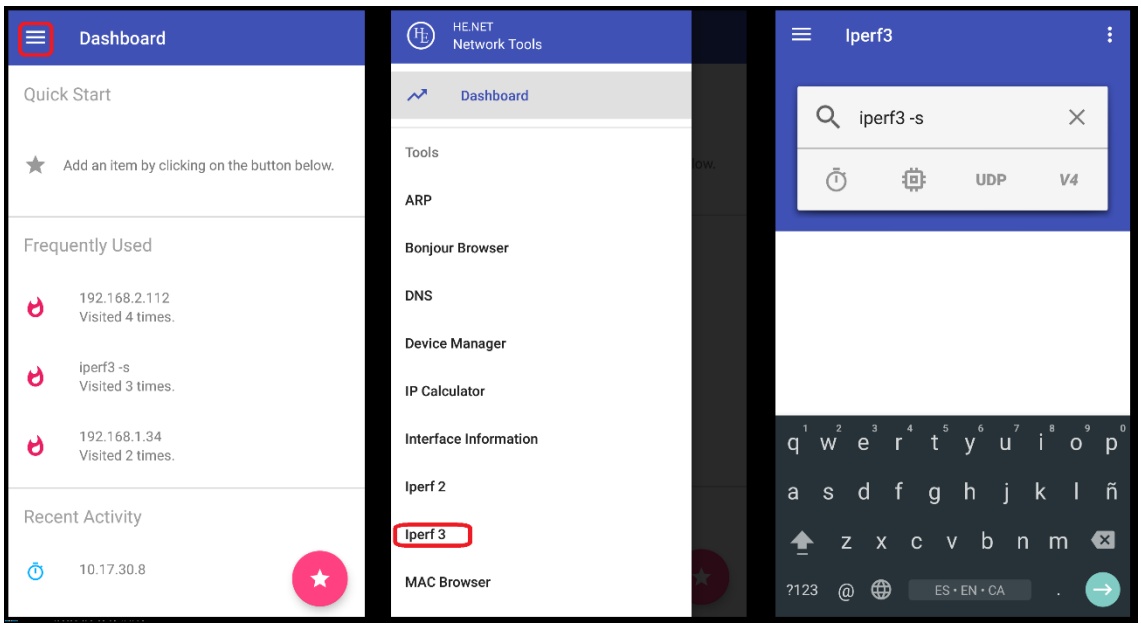


Figure 15: Network Tools application

d) 2 computers

The other 2 devices, which will act as a client in each network will be computers with a linux OS. These computers can have any kind of specifications. They will only need the Iperf3 tool installed. To do so, you will have to open a terminal and type:

```
apt-get install iperf3
```

Once installed, we make sure we started the servers in the tablets and we initiate the clients in the computers. To start them we open a terminal and we write the following command:

```
iperf3 -c "ip"
```

Where -c means, the device is acting as a client and sending packets to the server and *ip* is the IP address of the server where you want to send all the traces. In the client side you can also specify things such as whether you want to send TCP or UDP, the time you want the Iperf running or the transmission rate of the packets.

4.2. Scenarios

To test our tool and check that the results are satisfactory we have created different scenarios. The scenarios will have the same structure as in figure 16 but with different configurations to be able to see how the created program acts in different circumstances.

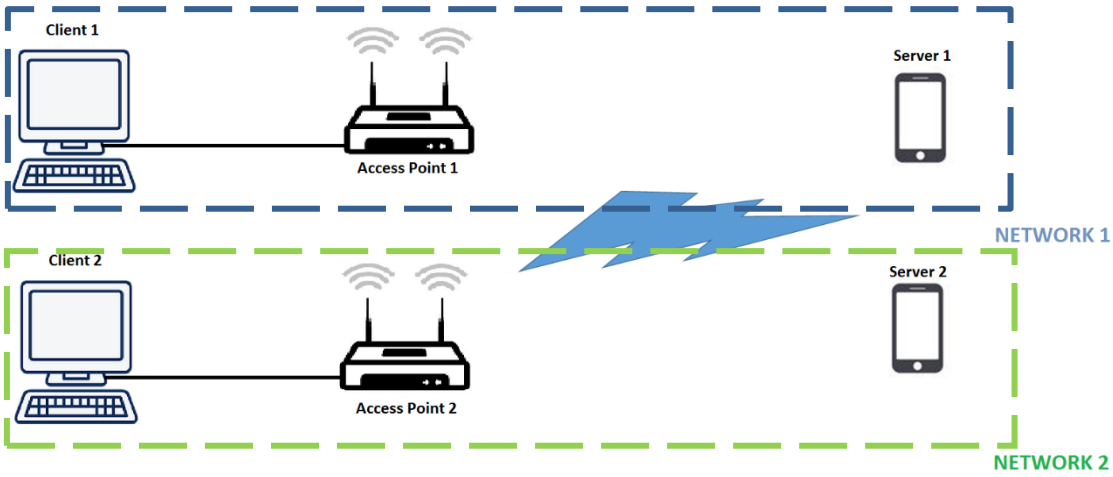


Figure 16: Scenario wireless network structure

As we can see, each network is formed by a computer, an access point and an android tablet. The computers will be connected by Ethernet to their respective APs and the tablets will connect via radiofrequency. This means both networks must share the medium, connected to the same channel, in order to see if they are interfering between them.

As said before, the access points will have a maximum transmission rate of 1 Mbps and a basic rate of 1-2 Mbps so it is easier to congest their wireless traffic and, therefore, obtain good results. In addition, both APs have to be working in the same channel.

4.2.1. UDP

In these scenarios, we are going to start both networks, by sending UDP packets to the server during 100 seconds. We are also going to specify transmission rates of 50, 100, 200, 300 and 500 Kbps, where Network 1 will start sending traffic 15 seconds before to see if its resources change when network 2 incorporates. To specify all these parameters in Iperf, you will have to write the next command in the terminal.

```
iperf3 -c "ip" -u -t 100 -b "transmission rate"m
```

Where *-u* specifies you want to send UDP packets, *-t 100* specifies the time in seconds iperf is going to send packets and *transmission rate* (followed by an *m*) to specify the transmission rate you want in Mbps.

4.2.2. TCP

In these scenarios, we are going to start both networks, by sending TCP packets to the servers during 100 seconds. In this case we can not specify the transmission rate the packets are transmitted. There will be two different scenarios. In the first one, Network 1 will start sending traffic 15 seconds before Network 2 and, in the second one, both networks will start at the same time. To send this type of packets we have to write the next command in terminal.

```
iperf3 -c "ip"
```

Where you only have to specify the server IP address because the tool sends as default TCP packets

4.2.3. UDP & TCP

In these scenarios, Networks 1 and 2 will send TCP and UDP packets respectively. There will be two different scenarios in which Network 1 (TCP) will start 15 seconds before and another one in which Network 2 will start 15 seconds before. This is done like this in order to see if, after the incorporation of a second network in their medium, their resources are being affected. In this case, Network 2, in charge of the UDP packets, will transmit the packets in a transmission rate of 1 Mbps.

4.3. Validation

In this section, we are going to analyse and determine if, with the results obtained, we can see any correlations and APs with traffic problems. As said in the previous section we have 3 types of scenarios that we are going to use for the validation of the metrics.

4.3.1. UDP Scenarios

We can divide the results obtained from these scenarios in 3 sections. This is because the metrics and plots obtained are similar between some of them. There is going to be a section where the results show us a positive correlation, which means their resources are well distributed. The second section will be in a midpoint between the first and third sections and, finally, the third section, where we can appreciate that the resources are being maximized.

However, there are some metrics that are common for all sections and which we are going to explain before inquiring in each of them.

The captures have been running with a mean of 140 seconds (between 125 and 150 seconds).

In addition, we can see that AP 1 has a signal strength mean of -35 dBm whereas AP 2 has a signal strength of -30 dBm so we can say that the traces obtained will be quite reliable in terms of information obtained.

a) 50 & 100 Kbps

Looking at the tables our tool has created, we can observe that the traffic in the medium is not really relevant. Both APs have a data packet rate of 4 and 8 packets/second in each transmission rate. There is also a Management packet rate of 16 packets/s in both APs.

If we go into more details, we can observe their throughput is about 40 Kbps in the first case and 80 Kbps in the second. As we can see, the throughput is slightly smaller than what we wanted because our tool gets the metric in all the time the capture has lasted, whereas, when you specify it in the Iperf, it will only do it until it stops creating traffic.

In addition, the retransmitted packets rate in the first case is about a 3% and a 5% in the second case.

With this information, we can say that their traffic is not being altered since the throughput is similar at what we specified in Iperf and the retransmission rate is very low.

Finally, if we observe the plots from figure 17 and the correlation table obtained from our code, we can observe that there is not any relation between the APs. If we get the correlation between APs every 5 seconds we can observe the correlation is positive and that varies from 0,5 to 0,9, which means the traffic from one AP does not change depending of what the other AP does.

If we look at the plot from figure 17, AP 1 starts 15 seconds before but, when AP 2 incorporates to the medium, his traffic is not being affected. We can also observe that when AP 1 stops sending packets (second 100) AP 2 sends the same amount of traffic. In addition, the APs are not using more than a 25 % of the medium, which means there is still enough resources for traffic from another APs to be used.

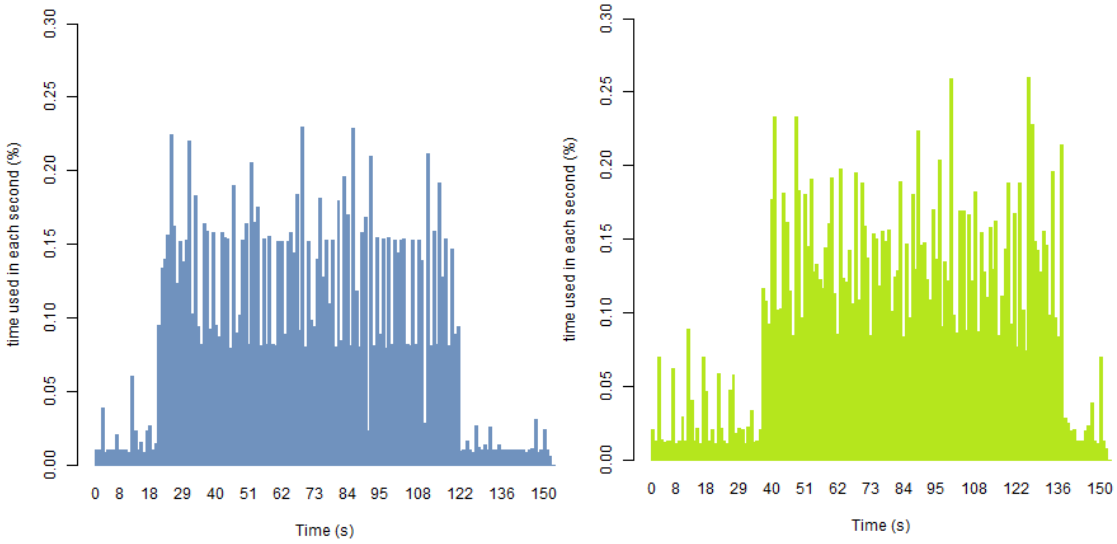


Figure 17: Traffic plot from AP 1 (Left) & AP 2 (Right) with 100 Kbps rate

With this results we can say that in transmission rates of up to 100 Kbps the traffic of the APs is normal and working correct and efficiently.

b) 200 & 300 Kbps

Looking at figure 18, we can observe that the traffic in the medium is starting to get more correlated between APs although is still irrelevant. Both APs have a data packet rate of 15 and 25 packets/second in each transmission rate. There is also a Management packet rate of 15 packets/s in both APs.

If we go into more details, we can observe their throughput is about 165 Kbps in the first case and 285 Kbps in the second. As we can see, the throughput is slightly smaller than what we wanted as explained in the previous section.

In addition, the retransmitted packets rate in the first case is about a 6% and an 8% in the second case.

With this information, we can see that the throughput still stands with what we specified but the retransmission rate is starting to get relevant.

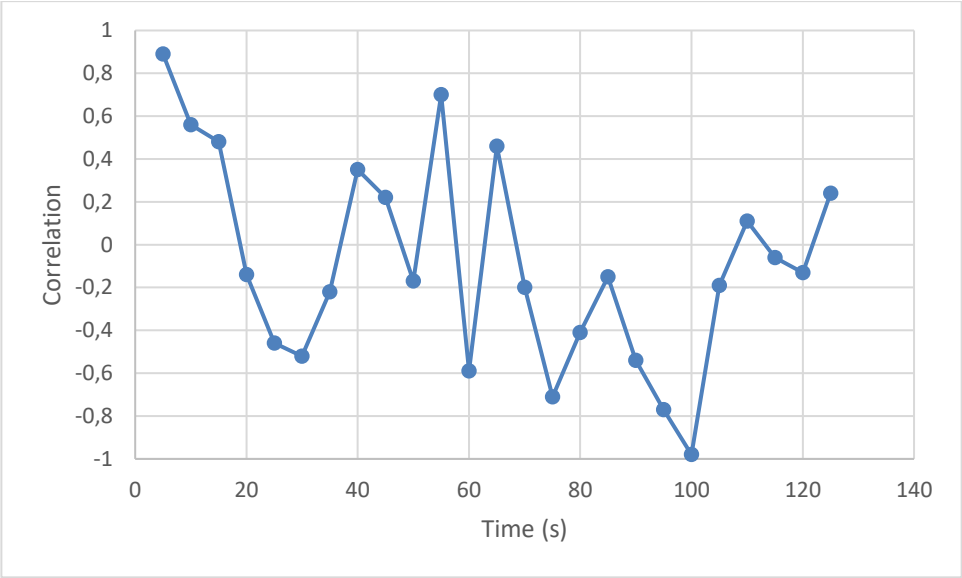


Figure 18: UDP correlation with a 300 Kbps rate

If we observe the plots from figure 18 and the correlation table obtained from our code, we can observe that the correlation is still insignificant in most of the time. If we get the correlation between APs every 5 seconds we can observe the coefficients are widely dispersed variating from 0,4 to -0,7, which means the traffic from one AP is getting more correlated with the other, although they still do not interfere between them. The only time where we can see the traffic is being affected is in time 100 with a negative correlation -0,98.

Finally, if we look at the plot from figure 19, AP 1 starts 15 seconds before but, when AP 2 incorporates to the medium, its traffic is not being affected. We can also observe that when AP 1 stops sending packets (second 100) AP 2 sends the same amount of traffic. Moreover, the traffic of both APs is in the barrier of the 40% which means there is not much free medium for a third AP to join without interfering in the others traffic.

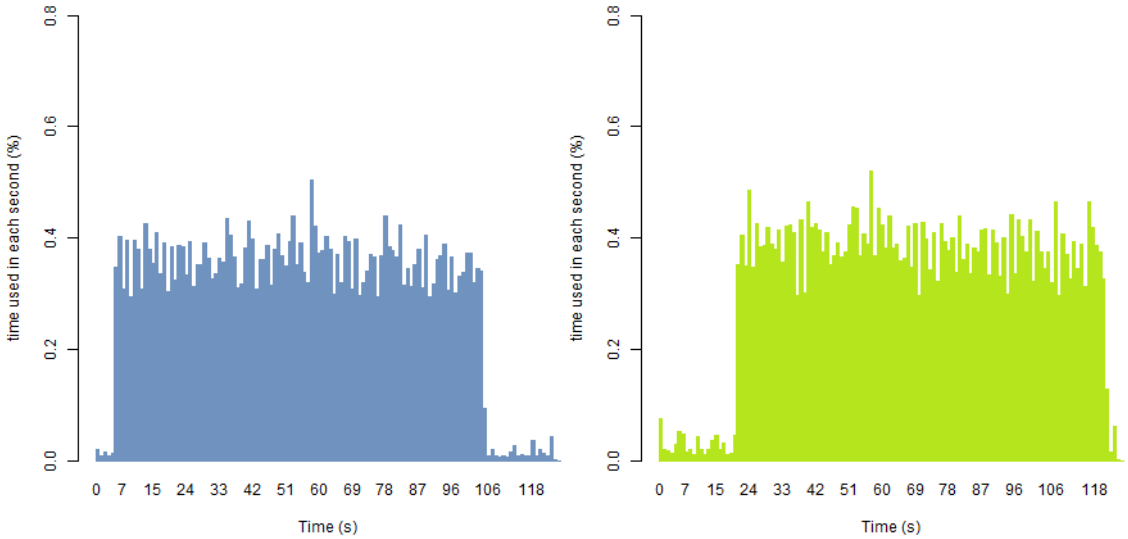


Figure 19: Traffic plot from AP 1 (Left) & AP 2 (Right) with 300 Kbps rate

With this results, we can say that in transmission rates of 300 Kbps the traffic is on the limit so they do not affect with other communications. Despite this, the individual traffic of each device still travels efficiently without being affected in their throughputs.

c) 400 & 500 Kbps

Looking at the tables our tool has created, we can observe that the traffic is being affected and that there are not enough resources to send all the packets in the transmission rate desired. Both APs have a data packet rate of 32 and 34 packets/second in each transmission rate. There is also a Management packet rate of 11 packets/s in both APs.

If we go into more details, we can observe their throughput is about 360 Kbps in the first case and 358 Kbps in the second. As we can see, the throughput has stalled in 360 Kbps which can mean it is the maximum transmission rate of the channel at this moment because of the congestion in it.

In addition, the retransmitted packets rate has increased to up to a 13% in both cases.

With this information, we can say that their traffic is getting saturated and that either, AP 1 and AP 2 are sharing the resources in the channel and, therefore, the channel traffic is full, or there is to many networks in this channel and, therefore, the resources are being shared with all of them.

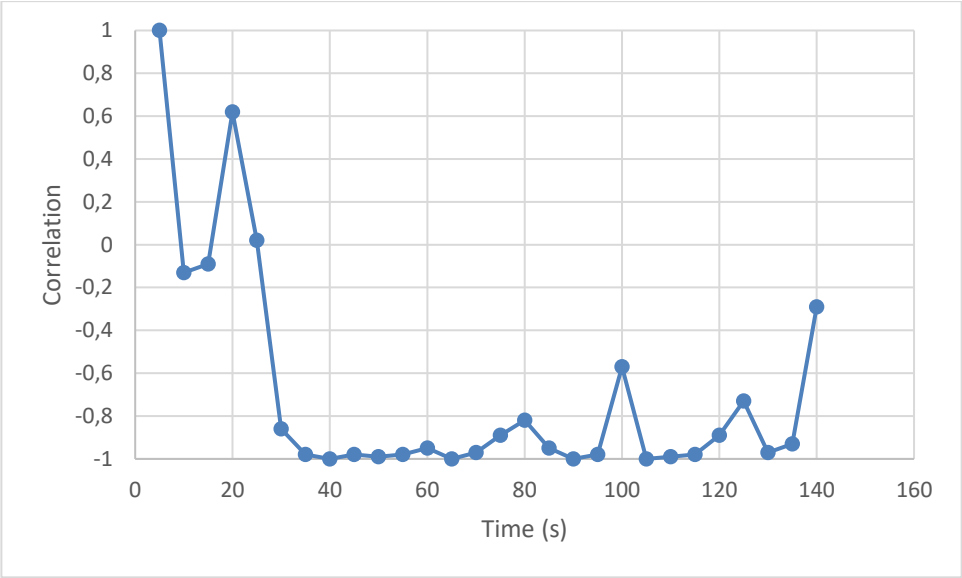


Figure 20: UDP correlation with a 500 Kbps

To leave doubts, we can observe the plots from figure 21 and the correlation obtained from our code in figure 20. If we get the correlation every 5 seconds, we can see that the value has increased considerably, changing to negative, and find them between -0,7 and -0,99. With this results we can say that both APs are draining the resources from the channel and that they are sharing all the available bandwidth leading them to have the maximum transmission rate which is below the desired.

If we look at the plot from figure 11, AP 1 starts 15 seconds before but, when AP 2 incorporates to the channel, its traffic decreases. At the time 45, AP 2 has a maximum peak using most of the channel whereas AP 1 has a minimum peak. At this point, we can see how AP 2 steals most of the channel resources for then going back to normal and share them evenly again. We can also observe that when AP 1 stops sending packets (second 115), AP 2 starts occupying most of the channel. With this plots we can see how both APs are trying to share the channel as fairly as possible. In addition, during most of the time they are using almost the 50% of the channel, which means all the resources are being used and, therefore, the channel is saturated.

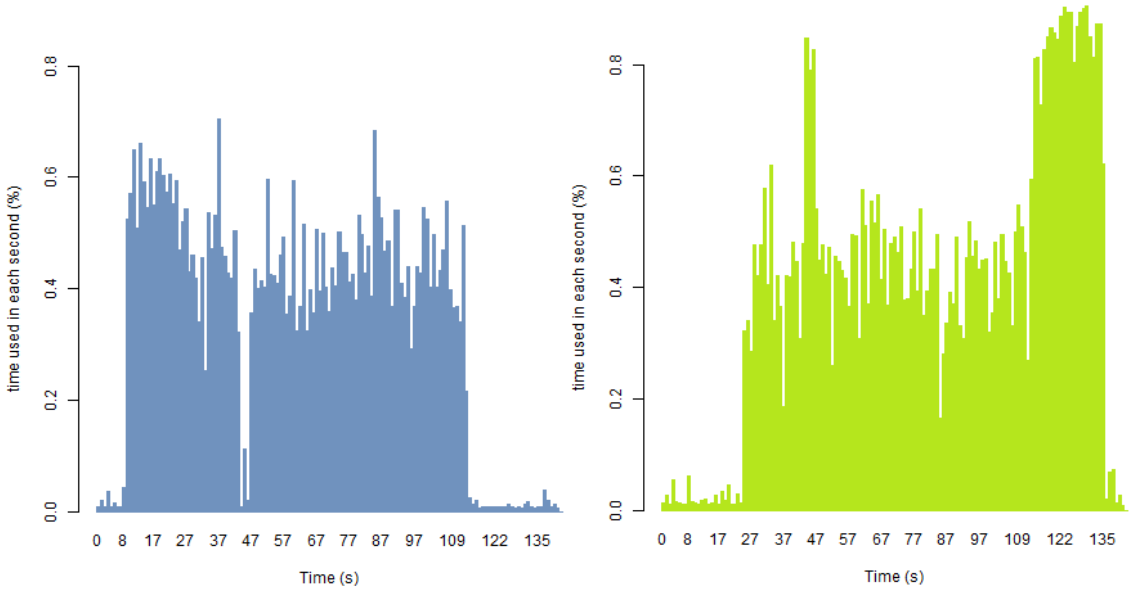


Figure 21: Traffic plot from AP 1 (Left) & AP 2 (Right) with 500 Kbps rate

With all this information, we can see that the maximum transmission rate at this moment is 360 Kbps and that there is a relation between the traffic of the APs. This means the traffic is not sent at the desired transmission rate because of the limited resources the APs have. Therefore we could conclude that 360 Kbps is the limit for 2 APs in the same channel if they want to send packets without interference between them.

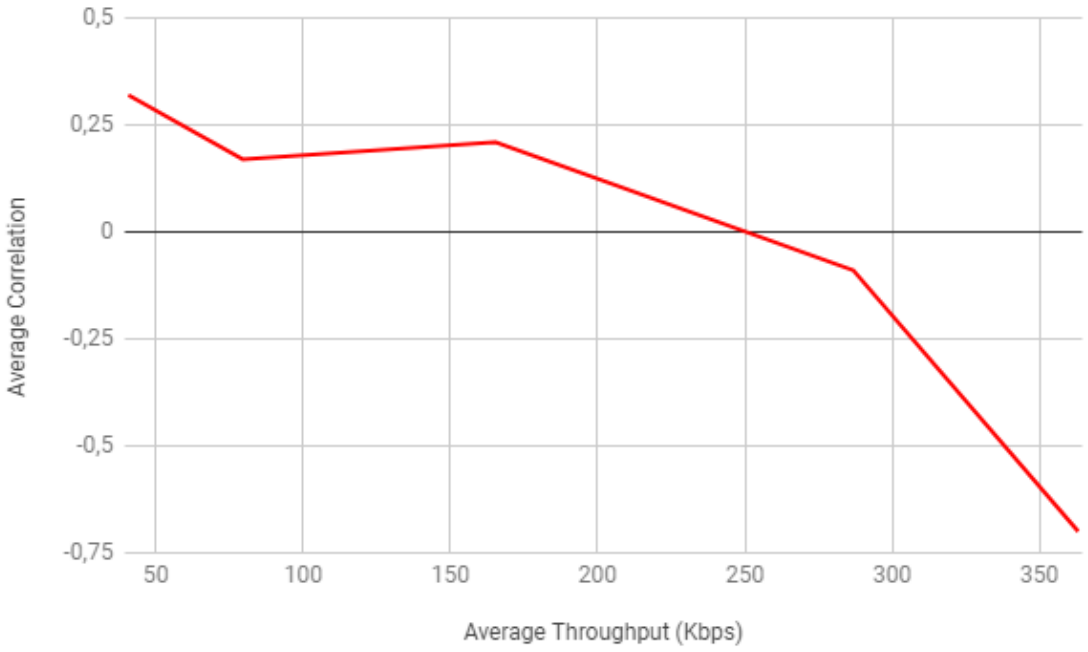


Figure 22: Average Correlation vs Transmission rate

With all the results obtained of the throughput and correlations means we have obtained this plot. In the graphic, you can see how, as the throughput increases, the correlation negatively increases. This means that the more bandwidth the

APs use, the more negative relation they will have and, therefore, the more congested traffic they will have. We can also see that from 70 to 170 Kbps the APs correlation maintains stable. Then, the correlation decreases considerably until -1. Having all these results we can say that the best transmissions rates for the APs using UDP, so they do not interfere between each other and have a reasonable transmission rate, are 70/170 Kbps.

4.3.2. TCP Scenarios

The running time of the captures has a mean of 160 seconds and in these scenarios, we can find common metrics that are going to be explained before entering to the sections. First of all, we can appreciate that AP 2 has a better signal, with -25 dBm, than AP 1, with -35 dBm. By looking at this, we can say that the sniffer is closer or that it receives less noise from AP 2 which means there are going to be less interferences although that the difference is not significant.

Finally, the throughput in both APs is of about 350 Kbps in both cases with a retransmission rate of a 20%. Considering the results from previous scenarios we can observe that the last rate has increased considerably. The increasing is because of the TCP packets used, that usually have more retransmissions due to its reliability, where all the packets must be sent correctly and without errors. Instead of dropping packets like UDP does, TCP will try to retransmit the packet until it is successfully send.

The scenarios will be divided in 2 sections. In the first section both APs will manage traffic at the same time, whereas in the second section, AP 1 will generate traffic 15 seconds before than AP 2 to see how the packet flux changes when there is congestion.

a) Both APs at the same time

Observing all the metrics we can see that the data packet rate of both APs is of 40 packets/second with a control packet rate of 21 packets/second.

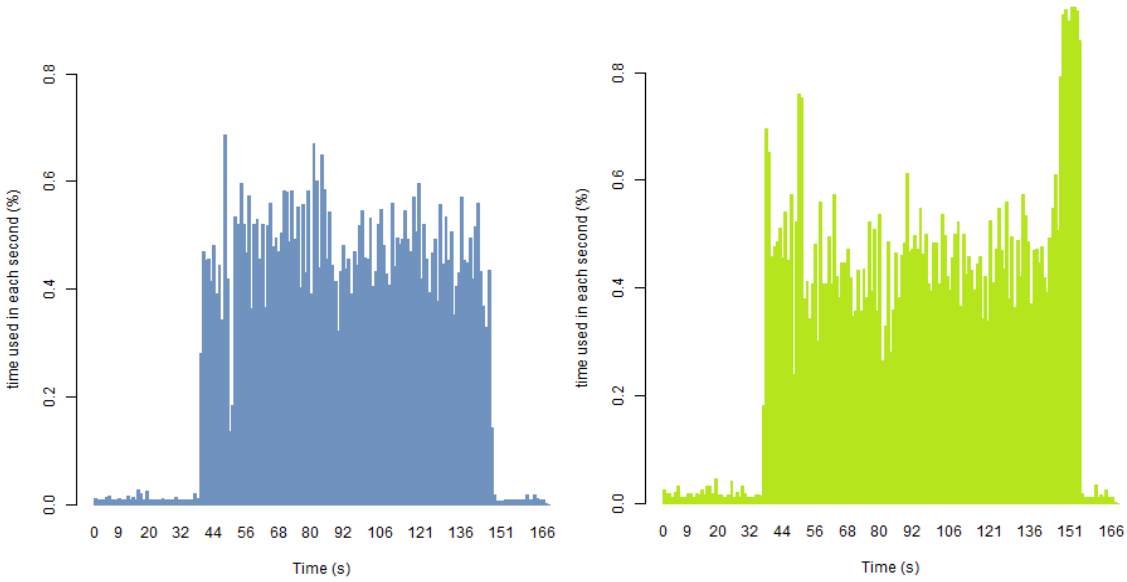


Figure 23: Traffic plot from AP 1 (Left) & AP 2 (Right) with TCP packets at the same time

If we focus on the correlation, we can observe that in the first seconds, where the APs are not transmitting any data packets, the correlations are positive and fluctuating around 0,8, which means the traffic generated between them does not affect their throughput.

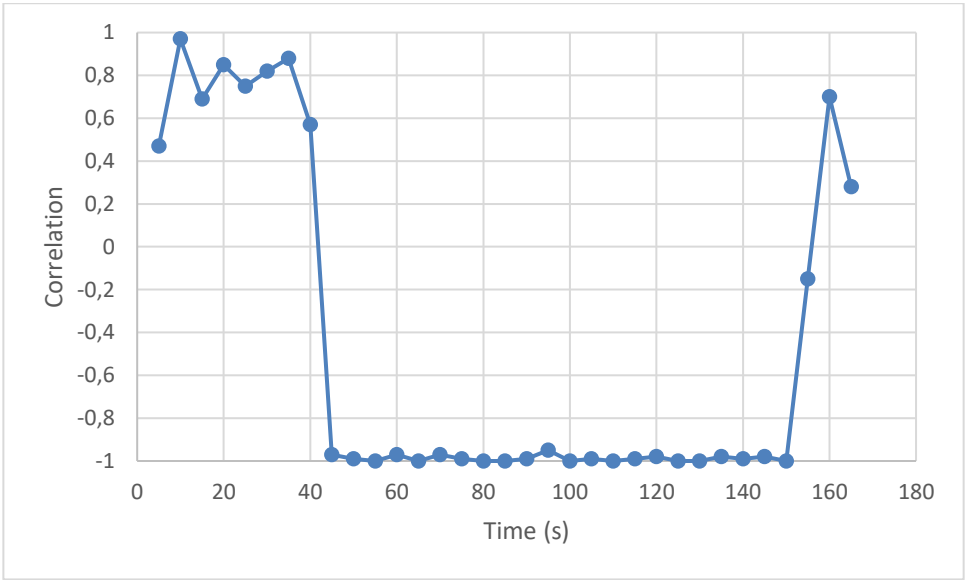


Figure 24: TCP correlation with both AP running at the same time

When we get to second 40, as seen in figure 23, the traffic starts in both APs and we can observe how the resources are being shared equally. From second 40 the correlation decreases and stays between -0,97 and -0,99, which means the APs are consuming most of the resources in the channel and are trying to share the medium. The situation remains until we get to second 147 where the correlations still stand between the values said before but the traffic varies. AP 2 starts consuming most of the resources and, therefore AP 1 is forced to wait until the situation is normalized. When both

APs stop sending we can observe how correlations go back to values as at the beginning.

b) AP 1 starts 15 seconds before

In this case, we can appreciate how the management and control packet rates slightly increase to up to 45 and 24 packets/second respectively. The increase can be caused due to the 15 seconds of difference between them. This allows each AP to use all the resources during those seconds, leading to the increase of data packets with their correspondent control packets.

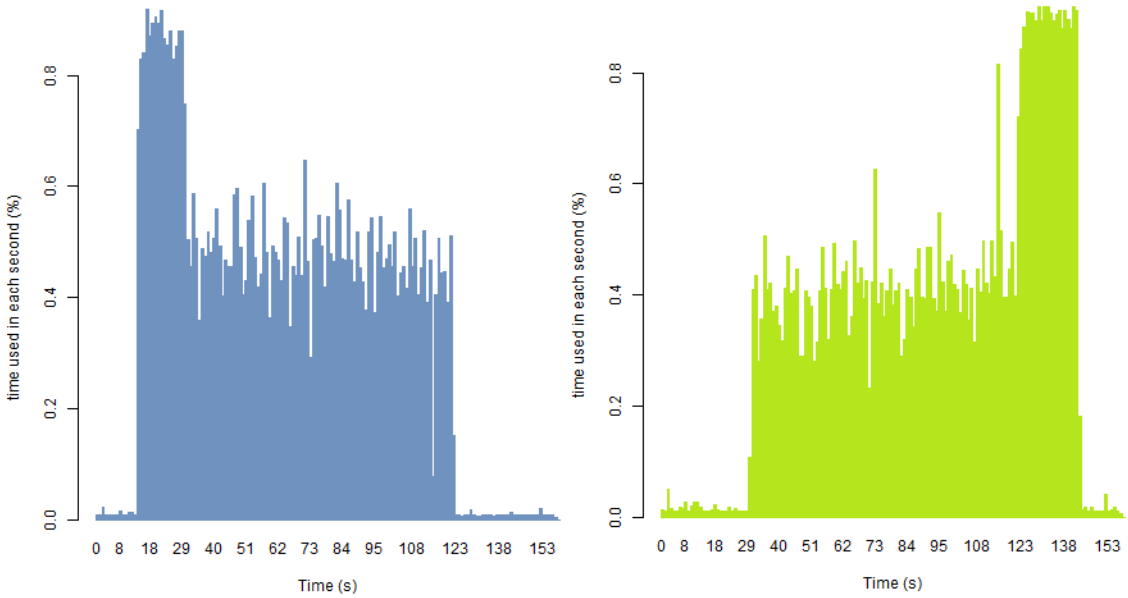


Figure 25: Traffic plot from AP 1 (Left) running 15 seconds before & AP 2 (Right)

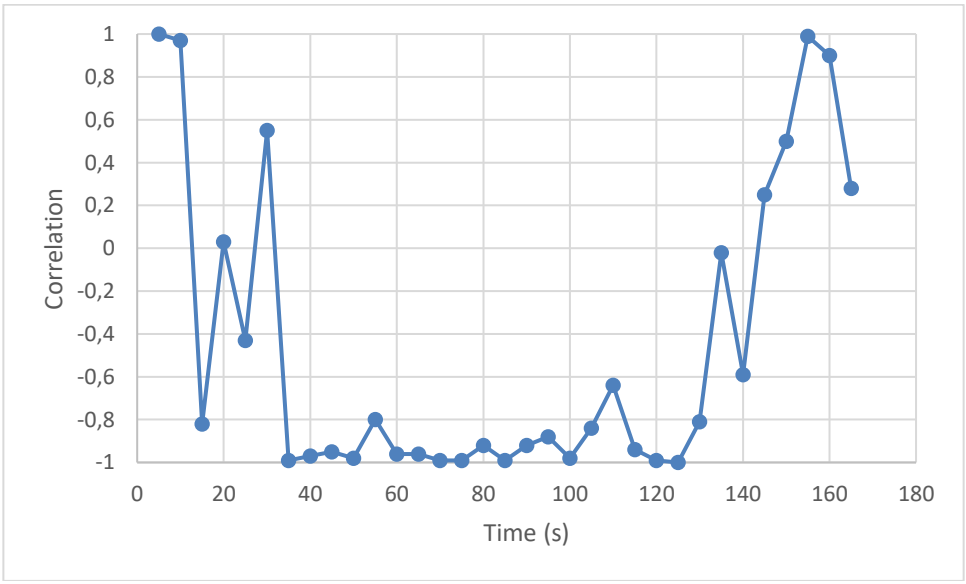


Figure 26: TCP correlation with AP 1 running 15 seconds before

If we focus on the correlation in figure 26, we can observe that in the first seconds, where the APs are not transmitting any data packets, the correlations are positive and fluctuating around 0,95, which means the traffic

generated between them does not affect their throughput. When we get to second 15, as seen in figure 25, the traffic starts in AP 1 but in AP 2 looks like the traffic does not change. Although AP 2 traffic looks like it is not changing much, we can observe by looking at the correlation at this point (between 15 and 30 seconds) that the values started to fluctuate from 0 to -0,6, leaving clear that there is a certain dependency on AP 1.

From second 30 AP 2 traffic joins the channel and we can observe how AP 1 starts to regulate its traffic in order to share the resources. At this point the correlation values are around 0,95.

Finally, when AP 1 stops transmitting packets at second 115, AP 2 takes advantage of the situation and consumes all the resources in the channel. One of the things we can not see in the plot at this point, is how the correlation is acting. The correlation values increase to between 0 and -0,5, just like it did at the beginning when only AP 1 was transmitting. When AP 2 stops sending packets the correlation stays in values of 0,9.

4.3.3. UDP & TCP Scenarios

The particularity of these scenarios is that each AP will treat with different kind of packets. AP 1 and AP 2 will be in charge of transmitting TCP and UDP packets respectively.

The time captured in each of the scenarios has a mean of 150 seconds. During this time, we can see that the management packet rate is of about 10 packets/second independently of the AP. We can also observe that AP 1 has a retransmission rate of 20% whereas AP 2 has a 15%. AP 1 has more retransmissions as expected because of the TCP packets.

The scenarios will be divided in 2 sections. The first section AP 1 will manage traffic 15 seconds before and, in section two, AP 2 will generate traffic 15 seconds before than AP 1 to see how the packet flux variate depending in which starts before.

a) AP 1 starting

During this capture, we can observe that AP1 has a data packet rate of 45 whereas AP 2 has 33. This means that AP 1 has send more data packets than AP 2 in the same time of transmission. In addition, although their packets rates are different, we can observe that their throughputs are of 360 Kbps approximately.

Before starting analysing figure 23, remark, although we specified Iperf to run during 100 seconds with UDP, in the plots AP 2 is active more time than expected.

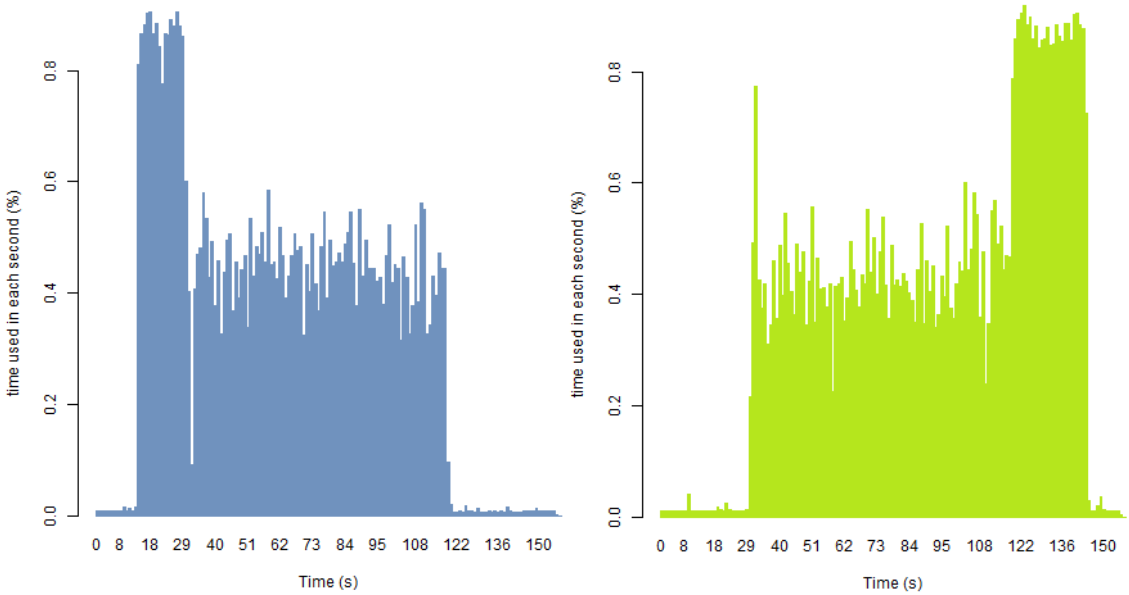


Figure 27: Traffic plot from AP 1 (Left) running 15 seconds before & AP 2 (Right) with UDP packets

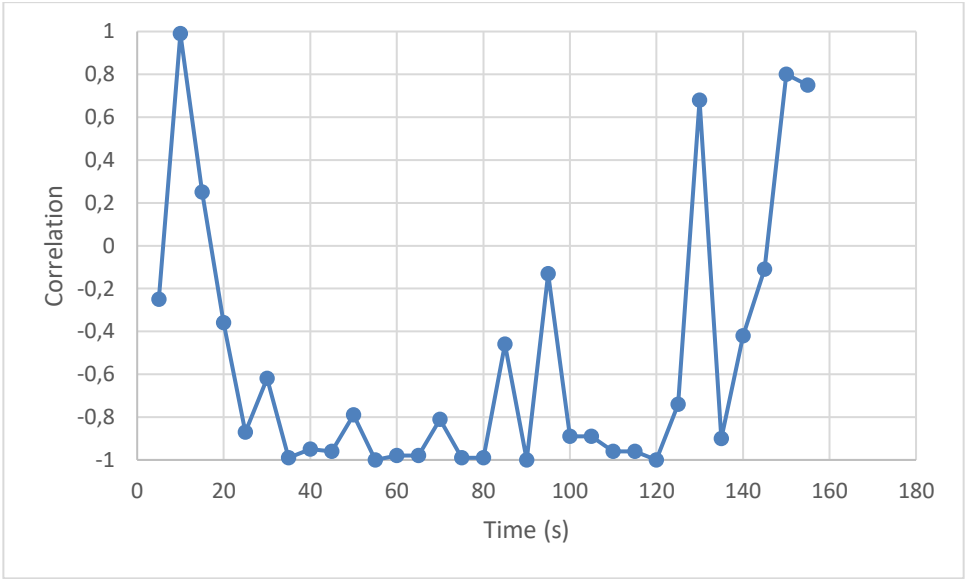


Figure 28: UDP & TCP correlation with AP 1 starting 15 seconds before

If we focus on the correlation in figure 28, we can observe an expected correlation at the beginning when none of the APs are transmitting. In second 15, AP 1 starts transmitting and occupying most of the channel. At this point, the correlation fluctuates from -0,35 and -0,80 meaning, although in the plots looks like the traffic is not affected, AP 1 is draining a part of AP 2 resources.

Once AP 2 joins the traffic in second 30, the traffic from AP 1 decreases and stays in similar values as AP2. In terms of correlation we can observe there is a big rate of dependency of about -0,98 in most of the period.

Finally, in second 125, by looking at the correlation, we can observe how AP 1 stops transmitting and AP 2 tries to exploit the channel until second 140.

During this point, correlation is fluctuating from -0,7 to 0, meaning AP 2 is using all the medium he can an draining AP 1 his resources.

b) AP 2 starting

During this capture, we can observe that AP1 have the same data packet rate of about 38 packets/second. In addition, we can observe there is a difference in their throughputs where AP 2 has 430 Kbps and AP 1 has 320 Kbps. Looks like AP 2 steals more resources than AP 1 in this scenario.

Before starting analysing figure 24, remark, although we specified Iperf to run during 100 seconds with UDP, in the plots AP 2 is active more time than expected.

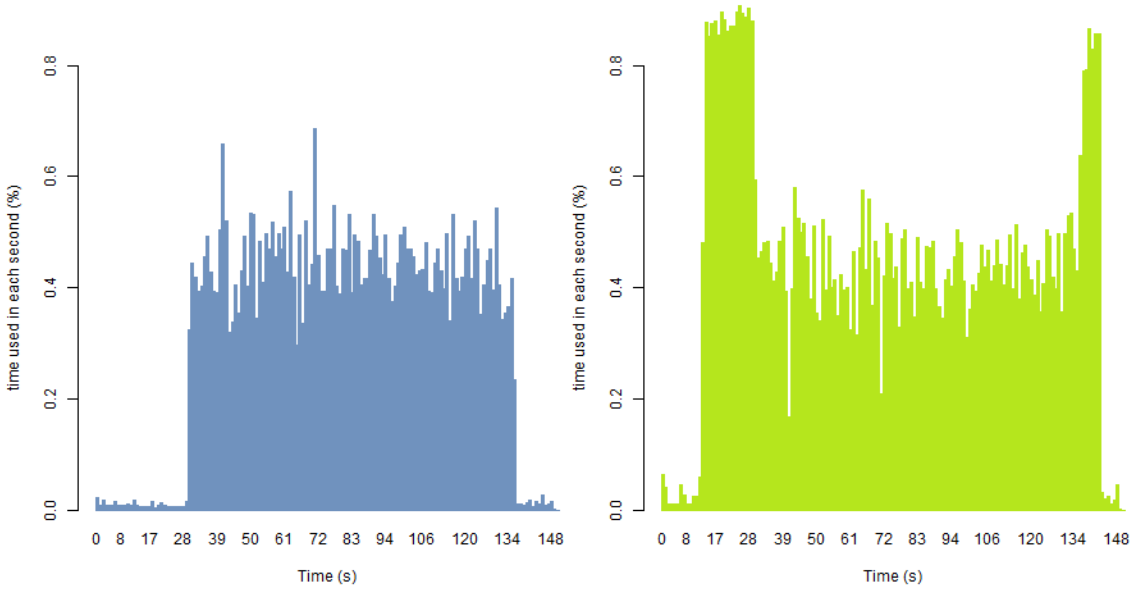


Figure 29: Traffic plot from AP 1 (Left) with TCP packets & AP 2 (Right) running 15 seconds before

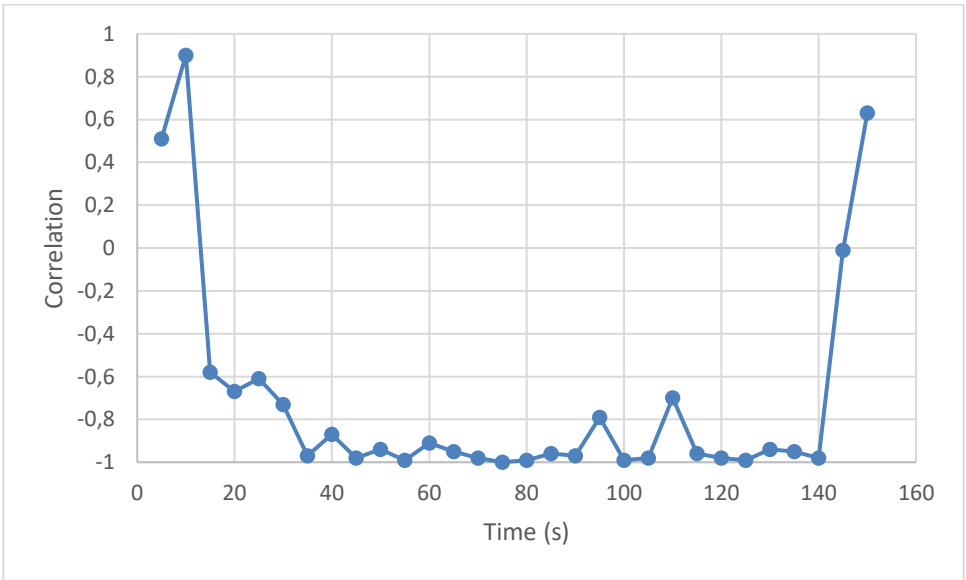


Figure 30: UDP & TCP correlation with AP 2 starting 15 seconds before

If we focus on the correlation in figure 30, we can observe an expected correlation at the beginning when none of the APs are transmitting. In second 15, AP 2 starts transmitting and occupying most of the channel. At this point, the correlation fluctuates from -0,6 and -0,7 meaning, although in the plots looks like the traffic is not affected, AP 2 is draining a part of AP 1 resources.

Once AP 1 joins the traffic in second 30, the traffic from AP 2 decreases and stays in similar values as AP1. In terms of correlation we can observe there is a big rate of dependency of about -0,97 in most of the period.

Finally, in second 145, by looking at the correlation, we can observe how AP 1 stops transmitting and AP 2 tries to exploit the channel until second 140. During this point, correlation is fluctuating from 0 to 0,6, meaning AP 2 is not using all the resources he could.

If we take a look into the throughput, once analysed the correlation and the plots, we can observe that AP 2 might have a bigger rate because it has been more time transmitting. In addition, this means AP 2 has been transmitting in moments where no other device was. All these factors can explain us the difference in throughput between both routers.

5. Analysing Real Scenarios

Once probed our software tool works correctly, we decided to implement the analysis of traces in real environments to investigate the existence of inter-AP dependencies in real WLAN deployments. In this part of the project, we sniffed the traffic from 3 places that seem to have quite a different network distribution. We will start from the most common situation you can find in a city to the most ideal situation.

The captures, unlike the lab experiments, have to sniff all the channels, since the amount of traffic and networks will be bigger. Therefore, we will use the script explained before specifying a changing time of 300 seconds so it goes through all the channels.

5.1. Apartment-Building Scenario

This capture was made in a usually common environment where probably most of the people will be. It was made in a first floor in a 6 floor building surrounded by other flats. In this kind of situations, you can find many different networks and it is perfect to see how the traffic works in such dense areas.

The sniffer has kept compiling traces during 4.200 seconds through all the 14 channels. Since there will be a lot of APs we decided to focus with the devices that give us interesting results.

5.1.1. Overall view

Once captured and analysed the traces by our tool, we can observe that the sniffer has detected traffic from 65 different APs.

Channel	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Nº APs	23	0	2	2	2	9	5	2	3	1	16	0	0	0

Table 4: Number of APs in each channel

As appreciated in table 4 we can observe that the main channels where more networks sniffed are channels 1, 6 and 11. The results coincide with the best channels to use for none overlapping. Most of the APs might be in those channels because of the configuration they have where, as a possible default setting, APs must only get assigned one of those three channels in order to avoid overlapping.

If we focus on channel 1, where most networks are, we can remark 3 APs, which we will called AP 1, 2 and 3. All the others are either inactive and sending only management frames, or with low throughput, traffic and retransmission rates.

	RSSI (dBm)	Manag. (packet/s)	Data (packet/s)	Throughput (Kbps)	Retrans. (%)
AP 1	-73	6.93	12.02	409,2	8,68
AP 2	-63	3,83	4,15	105,52	26,32
AP 3	-71	5,5	9,66	194,33	26,28

Table 8: The most used APs in cannel 1

We can observe that AP 1 is the one that has a bigger packet rate with a 409 Kbps rate of throughput. All this with an 8,68% of packet retransmission. Taking into consideration all the metrics from the table, sounds like AP 1 is draining most of the resources and, therefore, AP 2 and AP 3 have a bigger retransmission rate with a lower throughput. On the other hand, by looking at the low rates of transmission, we can also say that the correlation may not be relevant. To be able to decide whether they depending on AP 1 or not we have to look to their correlations.

5.1.2. Correlations

Our program did not find any significant correlation between APs and, if we check the correlation tables, we can see that there are hardly negative correlations between AP 2 and AP 3. However, although the correlation between AP 1 and AP 2 is not really negative, we can find that their relations tend to have a coefficient lower than 0 just as shown in table 9. Since the amount of time captured is quite high, we will calculate the correlation every 10 seconds.

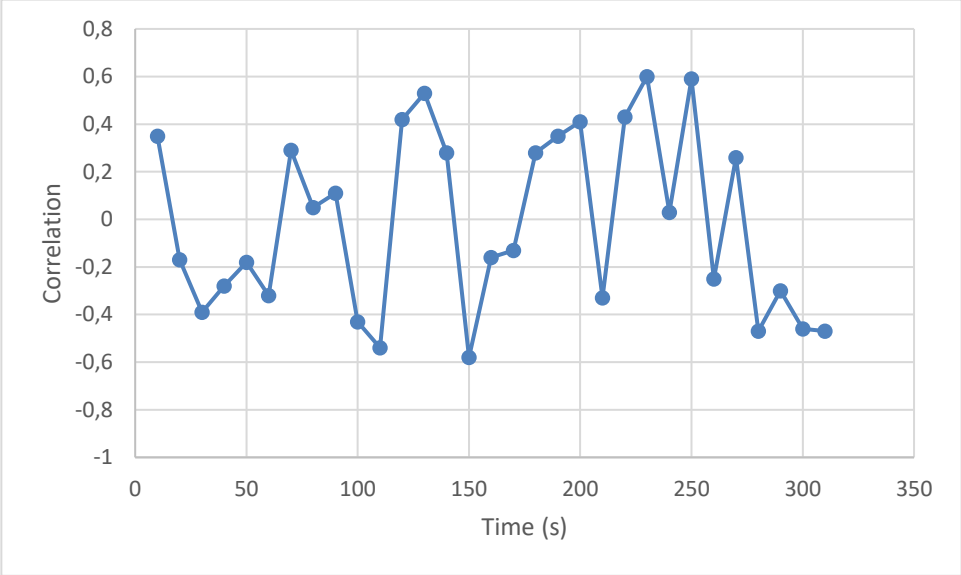


Figure 32: Correlation between AP 1 & AP 2

As figure 32 shows, the correlation fluctuates from -0,5 and 0,5 telling us that at some points there is a low relation between them but still not influentially enough to say that the resources from each AP are being corrupted from other APs. The high retransmission rates from AP 2 and AP 3 could be due to the position of the STAs regarding the AP. They might be connected but have a very low signal strength, fact that might generate the high rate of retransmissions.

Another theory could be because of the intervention of a third AP that is draining most of their resources but is not seen in the correlations because of the sniffer, who might be out of its scope.

Finally, as for the other APs in the other channels there is not any remarkable information that indicates their traffic is being affected for a third. In addition, you can tell the amount of traffic is not really high and maybe that is why there are low correlation rates.

5.2. Education Institution Scenario

This capture was made in a place where the installation of APs is very important. The network must cope with a big amount of stations and have a big scope to be able to grant an acceptable connection for all the users. In this case, the sniffer was inside a 2 floor building which a side faces with the street and the other with a square inside the university. In this situation, the amount of users should be more concentrated and, therefore, the information obtained should be interesting.

The sniffer has kept compiling traces during 4.230 seconds through all the 14 channels. Since there will be a lot of APs we decided to focus with the devices that give us interesting results.

5.2.1. Overall view

Once captured and analysed the traces by our tool, we can observe that the sniffer has detected traffic from 101 different APs.

Channel	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Nº APs	40	0	1	1	0	24	0	0	0	1	30	1	3	0

Table 5: Number of APs in each channel

As we can see the distribution of the APs is well organized where you can mostly find all the APs in the 3 non-overlapping channels 1, 6 and 11. The institution has chosen those to avoid the overlapping inside the installations. The aim of this is to create a network and avoid interferences from APs next to them. To do so, they need multiple APs and, in order to avoid disconnections when a user passes from an AP to another, the scope of each router needs to overlap with the others. Therefore, the APs next to them will be in another of the 3 nonoverlapping channels, due to avoid interferences just as figure 33 shows.

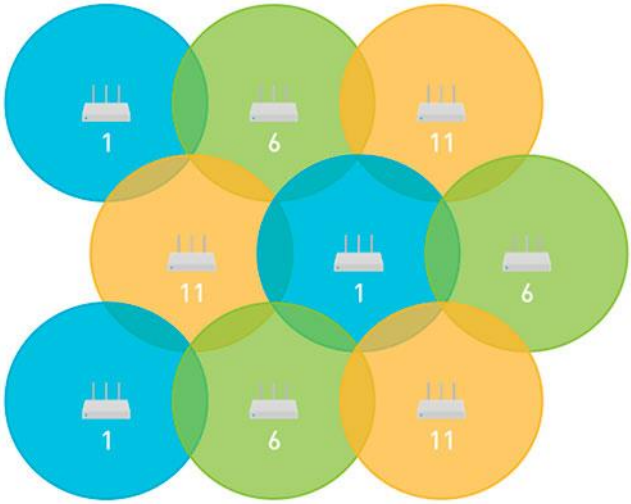


Figure 33: Distribution to avoid cannel interferences[10]

By looking at this configuration, we can see through the traffic plots that the remains from each channel are negligible. The remains are packets that do not correspond to the channel but due to overlapping we can find them in other frequencies.

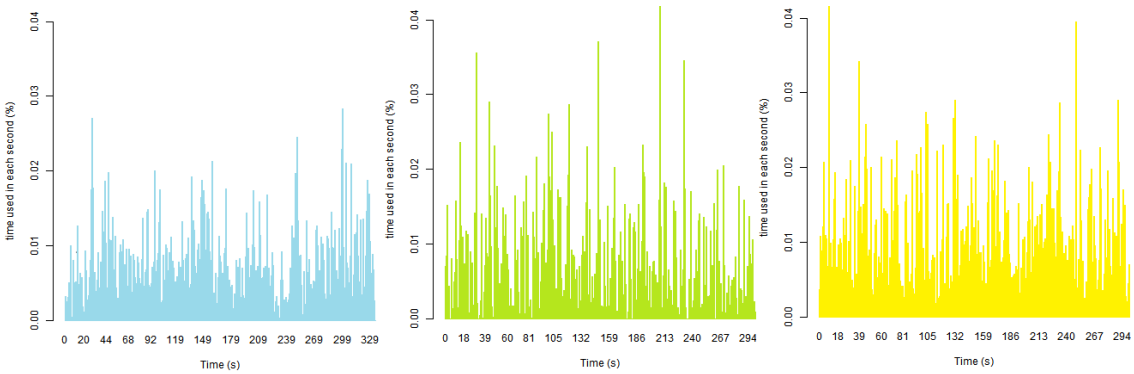


Figure 26: Remain packets in channels 1, 6 and 11 respectively

As seen, the remains do not overcome the 4% of the transmissions at any moment.

[10] Retrieved May 30, 2017 from http://cioperu.pe/revista_recursos/Fotos2015/Octubre/Informe_WiFi_6.jpg

5.2.2. Correlations

Analysing all the channels our program has only detected a relation that affects the traffic of others in channel 11 where there are 6 APs that having a negative correlation with the same AP.

	RSSI (dBm)	Manag. (packet/s)	Data (packet/s)	Throughput (Kbps)	Retrans. (%)
AP 1	-81	0,8	0,81	2,22	3
AP 2	-55	3,81	11,2	913	11
AP 3	-82	2,43	0,2	1	0,01
AP 4	-86	1,89	0	0	0
AP 5	-81	1,16	0	0	0
AP 6	-73	1,16	0	0	0
AP 7	-86	0,96	0	0	0

Table 6: The APs correlated with AP 2

As seen, the most active AP in the channel is AP 2 with a data rate of 11,3 packets/s and a transmission rate of 913 Kbps. By looking at the rest of APs there is not any anomaly that tells you there is something wrong with their traffic. AP 1 and 3 have a really low throughput and AP 4, 5, 6 and 7 are only using the channel to transmit management packets.

Therefore, by only looking at this metrics we can say that the causer of the negative correlation might be AP 2. However, we are going to check their correlation tables to see how it is affecting.

In this section, we are going to focus with the AP who has a stronger negative correlation. Anyhow, the tables and plots from the other APs will be in the appendix **A2. Plots & correlation tables from the rest of APs** to be able to see how AP 2 is affecting to the rest of APs.

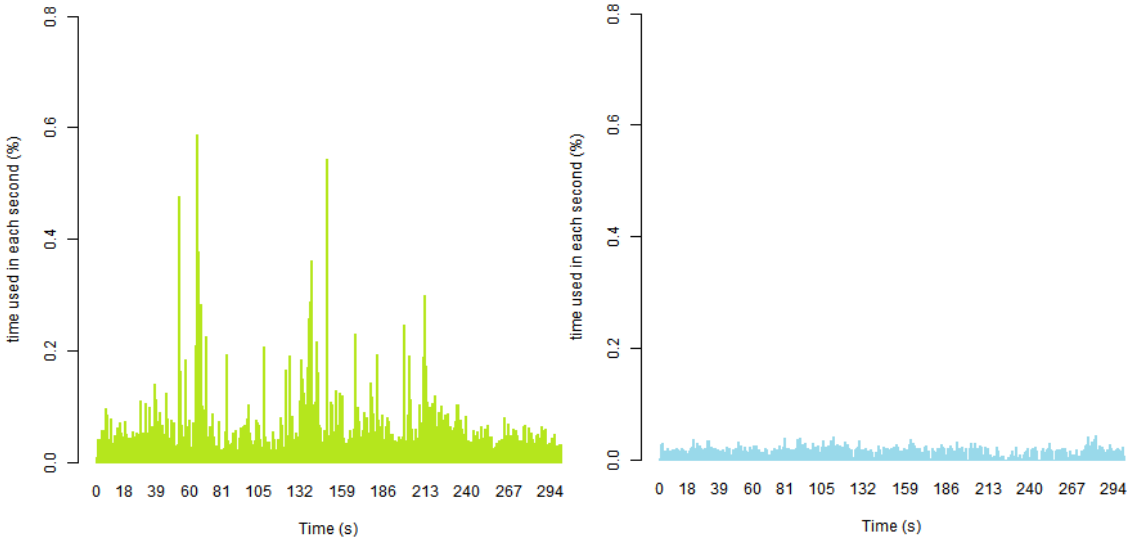


Figure 34: Traffic plot from AP 2 & AP 3 respectively

The plots show us how AP 2 has a few peaks where he uses most of the resources of the channel. However, those peaks do not last much and go back to the border of 20%. On the other hand, we have AP 2 with a low rate traffic which makes it difficult to analyse their relation visually. Therefore, we are going to look at the correlation table.

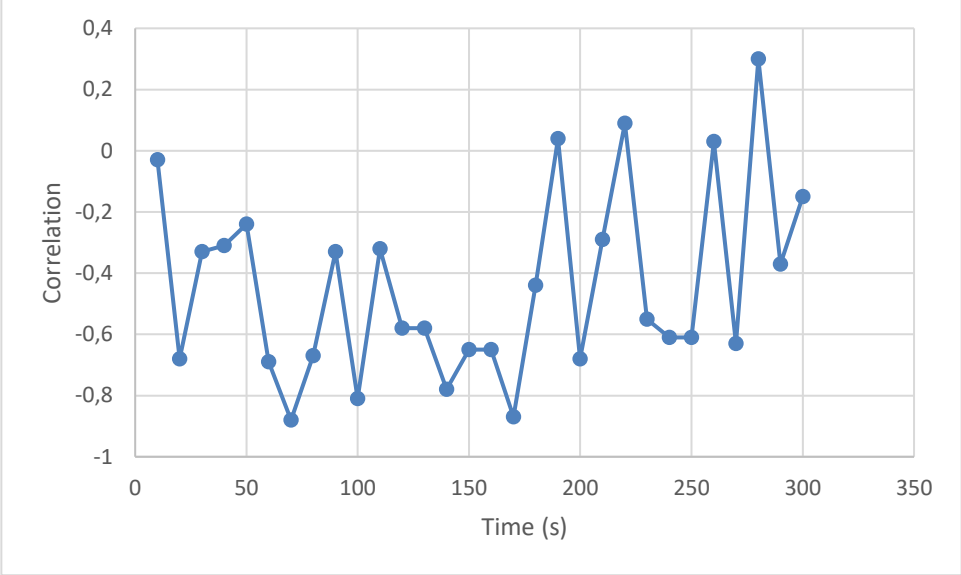


Figure 35: Correlation between AP 2 & AP 3

Relating the peaks seen in figure 32 with figure 35 we can observe that the correlation decreases to the border of -0,8. Those moments are in seconds 70 and 140. The rest of the time the coefficient maintains between 0 and 0,6. With all the results obtained, we can say AP 3 traffic is being affected because of the AP 2 utilization of the channel.

In addition, observing the activity and correlations of the other 5 APs we can say that AP 2 is monopolizing the channel at this moment and that it is the AP that more traffic density is generating.

5.3. Company Scenario

This capture was made in a place where the installation of network points is also very important. The network must cope with a big amount of stations and have a big scope to be able to grant an acceptable connection for all the users. In this case, the sniffer was inside a 16 floor building which makes the presence of external networks difficult. The election of this capture is to see how the channel should act in an ideal environment since all the APs detected belong and are configured by the company.

The sniffer has kept compiling traces during 2.000 seconds through all the 14 channels. Since there will be a lot of APs we decided to focus with the devices that give us interesting results.

5.3.1. Overall view

Once captured and analysed the traces by our tool, we can observe that the sniffer has detected traffic from 13 different APs.

Channel	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Nº APs	4	0	3	0	0	4	0	0	0	0	7	0	0	0

Table 7: Number of AP in each channel

As in the other cases, the most predominant channels used are channels 1, 6 and 11. We can also find an exception in channel 3 where there are 3 APs working. With this information, we can say there is not going to be interference due to overlapping channels. The installation following the same explanation explained in section 5.2.1. with figure 33 will reduce all the overlapping.

Since there is not any relation between APs, the packet and retransmissions rates are low and the amount of the devices is quite small, we are going to analyse the AP with more traffic in channel 11.

	RSSI (dBm)	Manag. (packet/s)	Data (packet/s)	Throughput (Mbps)	Retrans. (%)
AP 1	-47	3,83	3,71	0,036	7,3
AP 2	-47	5,84	17,85	1,146	7,4
AP 3	-47	5,48	2,27	0,037	10

Table 14: APs with more traffic in cannel 11

Inside this channels there are only 3 APs active during the capture and transmitting data packets. AP 2 is the device dealing with more transmissions with a data packet rate of 17,85 packets/ s and a transmission rate of 1,14 Mbps. Despite the big difference in terms of traffic, the retransmissions rates seem to be normal and there is nothing indicating an AP is being affected by a third.

5.3.2. Correlations

Comparing AP 2 traffic with any of the other APs, we can observe that he has a bigger packet flow occupying the 20% of the channel and with peaks of 30% and 40%. Despite does peaks, AP 3s traffic seems to not being affected at any moment of the capture.

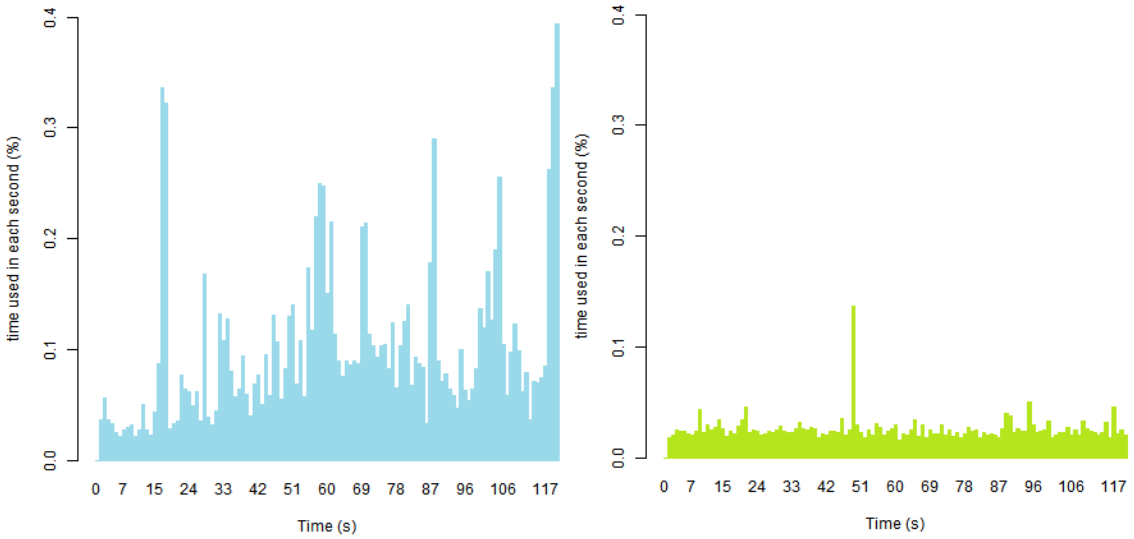


Figure 36: Traffic plot from AP 2 & AP 3 respectively

Our program did not find any significance correlations between APs and, if we check the correlation tables, we can see that there are hardly any negative correlations between AP 2 and AP 3.

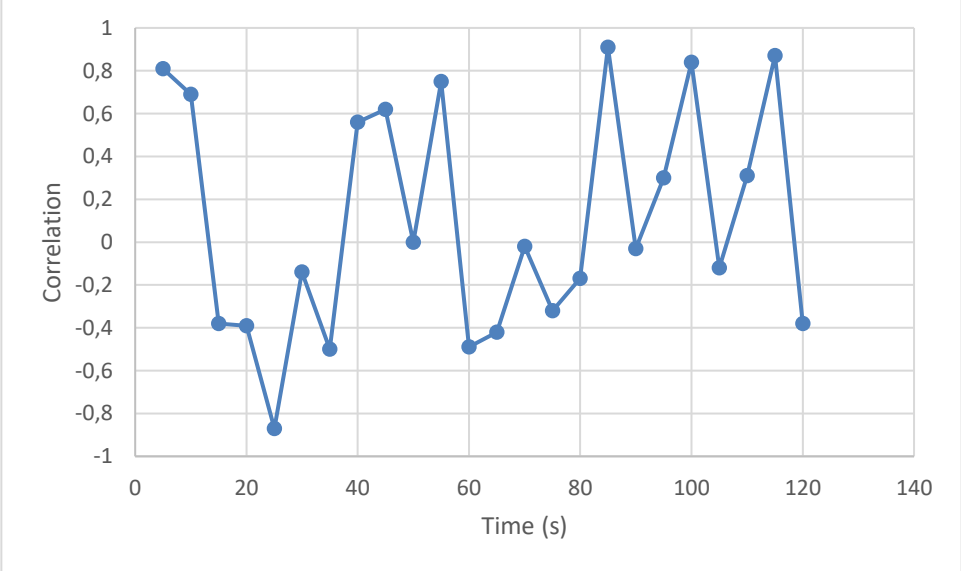


Figure 37: Correlation between AP 2 & AP 3

By looking at the results we can appreciate that the correlations do not follow any logical distribution. In addition, most of the time the coefficient tends to be positive, whereas the negative does not overpass the barrier of -0,5.

Considering all the information obtained, we can say that all the APs detected by the sniffer are not affected by thirds. The small amount of APs added with the distribution of the devices between the 3 nonoverlapping channels makes the situation perfect so that there is no unfairness when sharing the resources.

6. Conclusions

Communications have been in a constant evolution and, currently, they are moving to wireless technologies. This is causing the increase of traffic in the air medium. Therefore, because of the limited resources of the medium, the interferences in the channels between access points are raising.

To determine whether a channel is oversaturated we developed a wireless network analyser capable of obtaining all the useful information from the traces. Our main objective was to check if an access point was affected by some of his neighbours and how the throughput reacted in consequence. To do so, we looked at the correlation between each pair where, because of CSMA/CA utilization, an AP might wait for its neighbours to stop transmitting before sending something. This waiting generates the negative correlation we want to see to conclude if an AP is draining another's resources.

For this purpose, we first had to configure our sniffer to be able to capture 802.11 packets by changing the wireless adapter mode from promiscuous to monitor. This made possible sniffing all the traffic flying in the range of the computer instead of regular traffic from one point to another.

Once captured the traces correctly, after obtaining the *csv* file with all the information, we proceeded to analyse it with the tool created with R programing. To be able to confirm the metrics obtained were correct and conclusive, we proceeded to create different controlled scenarios. With the scenarios results we concluded satisfactorily that the software was working correctly and was capable of finding relevant correlations.

Finally, we used our tool in different real environments to see what kind of results there was and to make sure the traffic was well distributed in the areas captured. During the analysis, we saw that the only environment with significance correlations was in the Education Institution scenario. We also saw, that from the three spots, the most disorganized one was the Apartment-Building scenario. Moreover, to reduce the interferences from neighbours from other channels, with all the results analysed, we observed that most APs were in the three non-overlapping channels. We also concluded that the most ideal environment was the Company Scenario which, because of its height isolation, avoided any third APs not controlled by them. In this case, most of the devices were in those channels in order to reduce the interferences.

This project has been a major learning experience for me. I have learned in more detail how the wireless networks work and how they have to share their resources to avoid congestion. In addition, the R language was an unknown data manager for me and it showed me how efficient this language can be in order to analyse big amounts of data.

The next steps to continue with this project could be improving the code made by adding the obtainment of new useful metrics and adding a visual GUI interface, so it is easier for any user to use it without the need of having a programming knowledge. In addition, another step could be changing the APs configurations in function of the metrics analysed to improve their activity and see how they react.

References

[1] CCM. *What is WiFi and How Does it Work?*
http://ccm.net/faq/298-what-is-wifi-and-how-does-it-work#what-does-wifi-stand-for
(Retrieved November 25, 2016)

[2] Jerome Henry. Ciscopress. *WLAN Fundamentals.*
http://www.ciscopress.com/articles/article.asp?p=1876001
(Retrieved November 25, 2016)

[3] Daniel E.Capano. Control Engineering. *WLAN topologies.*
http://www.controleng.com/single-article/wlan-topologies/499aeba08a9b2d0741e5f992974db2ce.html
(Retrieved November 25, 2016)

[4] Intel. *Wi-Fi diferentes protocolos y velocidades de datos*
https://www.intel.la/content/www/xl/es/support/network-and-i-o/wireless-networking/000005725.html
(Retrieved December 20, 2016)

[5] Ian Poole. Radio-Electronics. *IEEE 802.11 Wi-Fi Standards*
http://www.radio-electronics.com/info/wireless/wi-fi/ieee-802-11-standards-tutorial.php
(Retrieved December 20, 2016)

[6] Ian Poole. Radio-Electronics. *Wi-Fi / WLAN Channels, Frequencies, Bands & Bandwidths*
http://www.radio-electronics.com/info/wireless/wi-fi/80211-channels-number-frequencies-bandwidth.php
(Retrieved December 20, 2016

[7] Pablo Brenner. Breezecom. *A Technical Tutorial on the IEEE 802.11 Protocol*
http://www.sss-mag.com/pdf/802_11tut.pdf
(Retrieved January 28, 2017)

[8] Wireshark. *FAQ*
https://www.wireshark.org/faq.html#q1.1
(Retrieved November 15, 2016)

[9] Syngress Books. *Wireless Sniffing with Wireshark, Chapter 6.*
http://cdn.ttgtmedia.com/searchNetworking/downloads/Orebaugh_Wireshark_Chapter_6.pdf
(Retrieved November 15, 2016)

[10] Kent State University. *SPSS Tutorials. Pearson Correlation.*
http://libguides.library.kent.edu/SPSS/PearsonCorr
(Retrieved April 5, 2017)

[11] R-Project. *What is R?*
https://www.r-project.org/about.html
(Retrieved November 15, 2016)

Appendix

A1. Hopping Channels Script

This is the changehop.sh code.

```
#!/bin/bash

IFACE=wlan0

bg="1 2 3 4 5 6 7 8 9 10 11"

bg_b="$bg 12 13 14"

a="36 40 44 48 52 56 60 64 149 153 157 161"

bga="$bg $a"

bga_b="$bg_b $a"


while true ; do

      for CHAN in $bg_b ; do

            echo "Switching to channel $CHAN"

            iwconfig $IFACE channel $CHAN

            sleep 30

      done

done
```

Where you can indicate the number in seconds you want the program to stay in a channel. You can also choose what channels you want to switch to by changing in the line 19 the value bg_b to any of the values in the top of the code.

To run the code you just have to type the following command

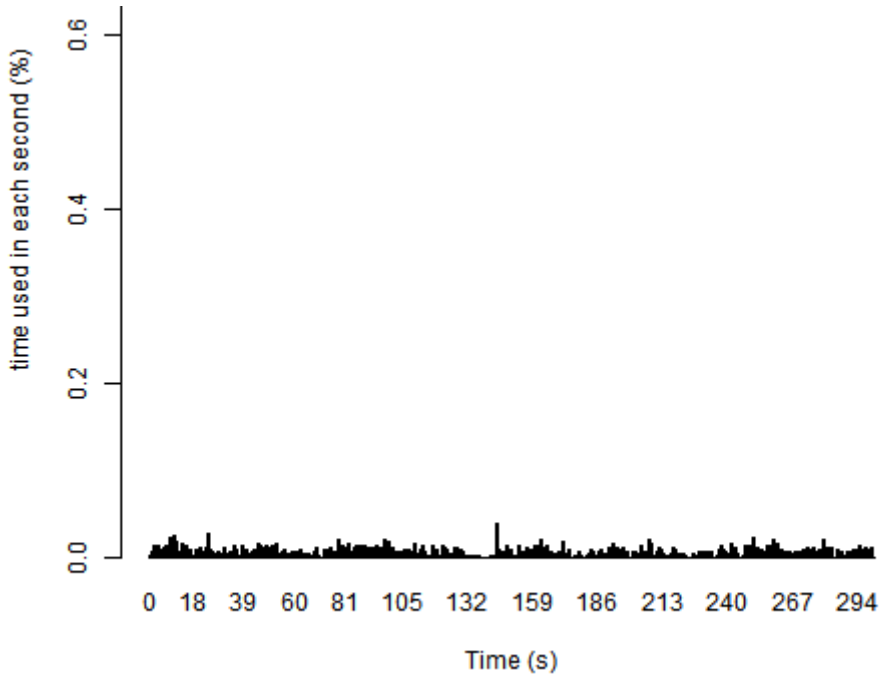
```
./"script"
```

Where script is the name you assigned to the script file.

A2. Plots & correlation tables from the rest of APs

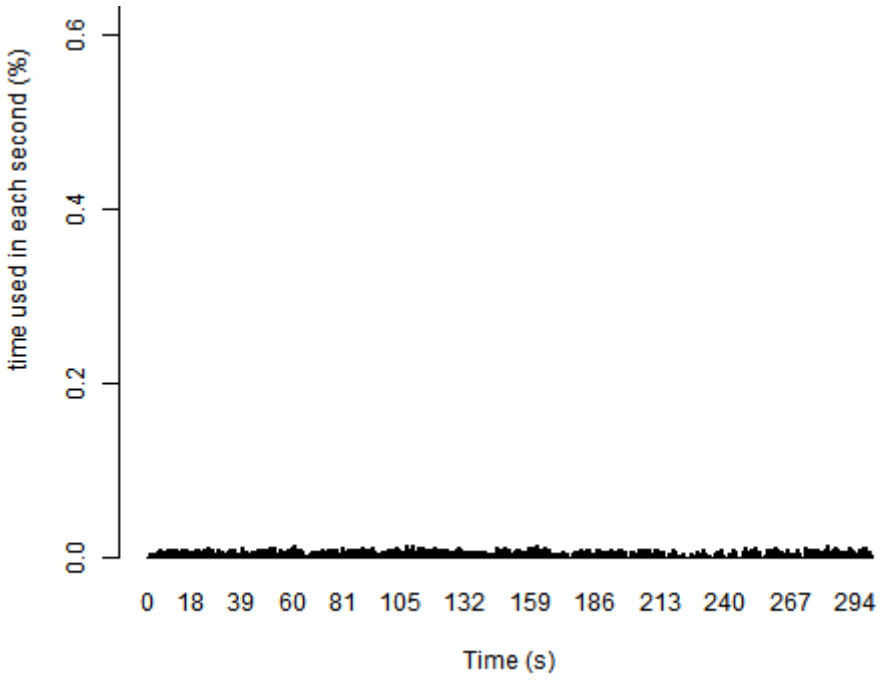
Remark the correlations obtained are only with AP 2 due to the low activity of the others. Therefore, the other correlations coefficients are positive and not relevant.

a) AP 1



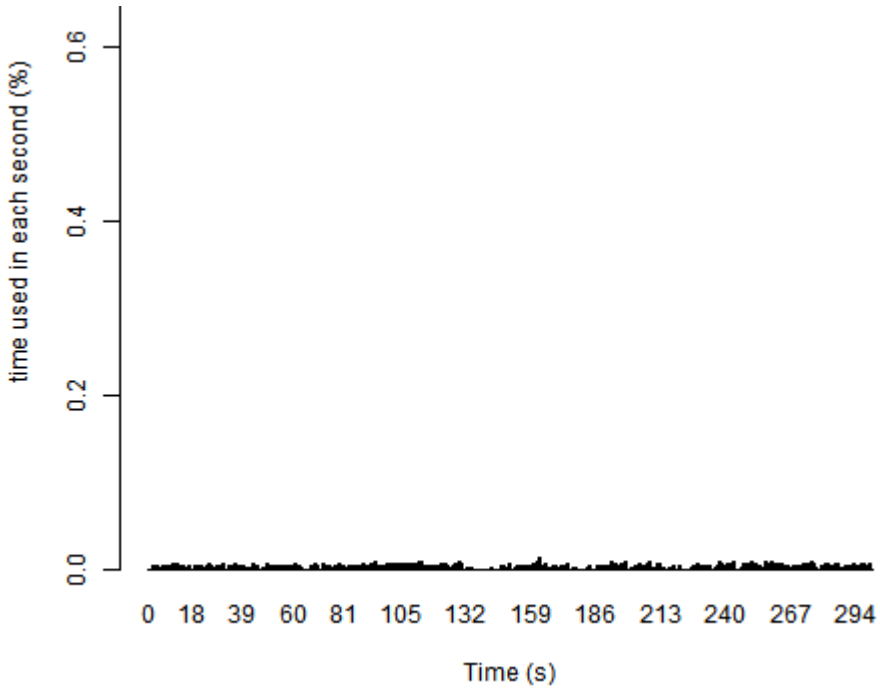
Time (s)	10	20	30	40	50	60	70	80	90	100	110	120
Correlation	0,22	-0,61	-0,14	-0,40	0,30	-0,76	-0,80	-0,60	-0,83	-0,04	-0,75	-0,09
Time (s)	130	140	150	160	170	180	190	200	210	220	230	240
Correlation	-0,17	-0,74	-0,36	-0,67	-0,62	-0,44	-0,28	-0,71	-0,57	0,37	-0,58	-0,65
Time (s)	250	260	270	280	290	300						
Correlation	-0,37	-0,39	-0,44	0,00	-0,59	-0,05						

b) AP 4



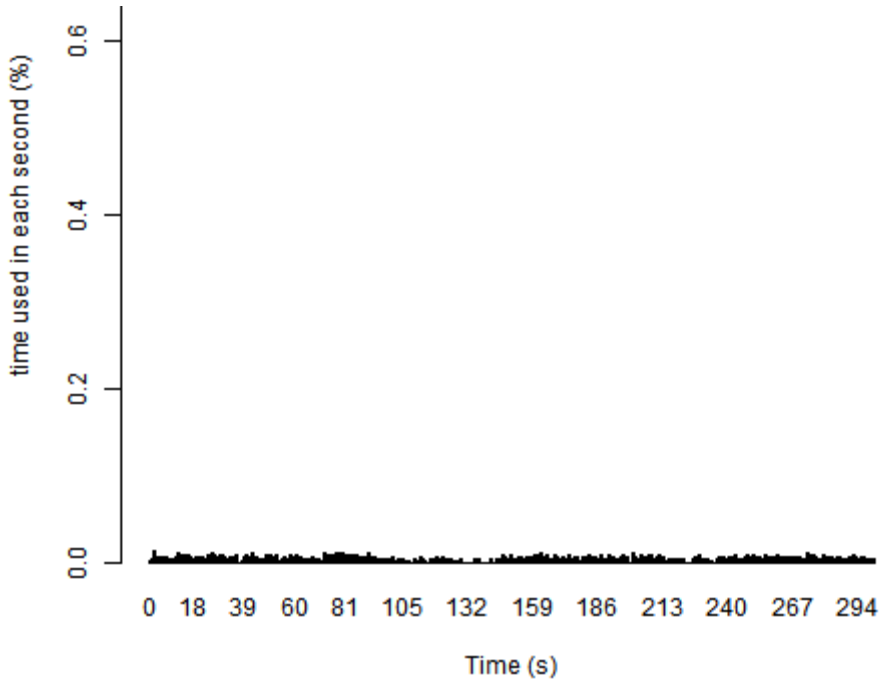
Time (s)	10	20	30	40	50	60	70	80	90	100	110	120
Correlation	0,18	-0,52	-0,74	-0,51	-0,56	-0,80	-0,89	-0,43	-0,53	-0,49	-0,58	0,36
Time (s)	130	140	150	160	170	180	190	200	210	220	230	240
Correlation	-0,39	-0,64	-0,63	-0,75	-0,81	0,01	-0,17	-0,87	-0,60	-0,07	-0,65	-0,80
Time (s)	250	260	270	280	290	300						
Correlation	-0,33	-0,02	-0,45	-0,37	0,17	-0,59						

c) AP 5



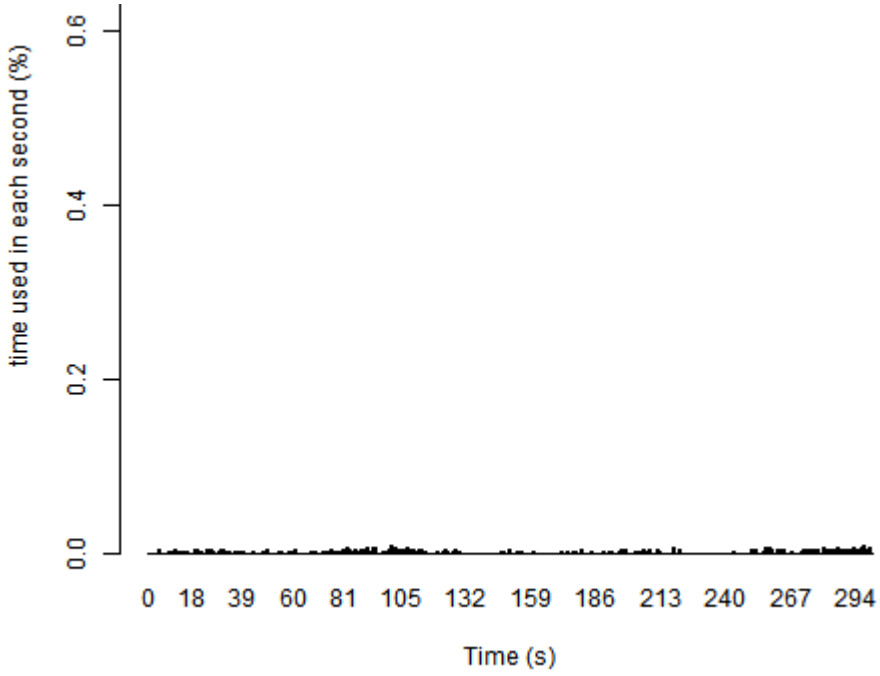
Time (s)	10	20	30	40	50	60	70	80	90	100	110	120
Correlation	0,43	-0,73	-0,27	-0,72	-0,50	-0,86	-0,46	-0,26	-0,49	-0,43	-0,72	-0,72
Time (s)	130	140	150	160	170	180	190	200	210	220	230	240
Correlation	-0,55	-0,53	-0,21	-0,69	-0,54	-0,53	-0,56	-0,73	-0,23	0,51	-0,86	-0,67
Time (s)	250	260	270	280	290	300						
Correlation	-0,55	-0,63	-0,45	-0,27	-0,28	-0,27						

d) AP 6



Time (s)	10	20	30	40	50	60	70	80	90	100	110	120
Correlation	-0,02	0,09	-0,16	-0,77	-0,26	-0,71	-0,64	-0,87	-0,08	-0,69	-0,39	0,09
Time (s)	130	140	150	160	170	180	190	200	210	220	230	240
Correlation	-0,07	-0,41	-0,47	-0,83	-0,63	-0,48	-0,01	-0,70	-0,61	0,09	-0,75	-0,79
Time (s)	250	260	270	280	290	300						
Correlation	-0,55	-0,21	-0,57	-0,24	-0,23	0,52						

e) AP 7



Time (s)	10	20	30	40	50	60	70	80	90	100	110	120
Correlation	0,06	-0,52	-0,52	-0,26	-0,49	-0,32	-0,44	-0,52	-0,30	-0,65	-0,43	-0,50
Time (s)	130	140	150	160	170	180	190	200	210	220	230	240
Correlation	-0,39	-0,57	-0,40	-0,31	-0,14	-0,21	-0,41	-0,47	-0,17	0,02	-0,49	-0,13
Time (s)	250	260	270	280	290	300						
Correlation	-0,46	-0,87	-0,21	-0,50	-0,53	-0,56						

A3. Program Code

```
#install.packages("plyr")
#we call the library plyr (installed previously)
library(plyr)

############################Function 1#############################

#In this function the variable input is the name of file you want to
import
#Make sure the name is between brackets
export_aps_data <- function(input){

  #creates a data frame with the input (in this case the capture
      exported in csv format from wireshark)
  capture <- read.table(input, header=TRUE,sep=",",
stringsAsFactors=FALSE)

  #creates a data frame containing only the beacons packets
  beacons <- subset(capture, Type.Subtype == "Beacon frame")

  #creates a data frame with all the aps detected in the capture by
      checking the beacon packets which are only
  # send by Access Points
  aps <- ddply(beacons, .(BSS.Id, SSID, Current.Channel), summarise,
RSSI=mean(Signal.strength..dBm.))
  aps<- aps[order(aps$SSID, decreasing=TRUE),]

  #returns a list of the capture, beacon and aps data frames
  return(list(capture= capture, beacon =beacons, aps=aps))

  }

############################Rssi#################################

#This function will be in charge of ploting all the Aps showing their
mean RSSI in a histogram.

#The ploting wil be saved in a png file in the same folder the program
is executing.
rssi <- function(){

  #You create a png file
  png(file="RSSI.png")

  #You plot the histogram inside the png file
  barplot( aps$RSSI, names.arg = aps$BSS.Id, xlab="AP", ylab="RSSI",
border="red", las=2)

  #You close the png file
  dev.off()

  }
```

```
##############################Function 2#################################

#This function is on charge of getting the information from each channel
you have been sniffing

channel_data <- function(capture){

  #Creates a data frame with 2 columns ( Channel and Time it has been
scaning)
  Channel.Data <-
data.frame(Channel=integer(),time.scanning=numeric(),stringsAsFactors=FAL
SE)

  #initiate provisional variables to be able to calculate later the
      amount of time you
  # have been sniffing the channel
  firstt=0.0000
  lastt=0.0000
  time=0.0000
  channel=0

  #A loop that goes through all the capture table
  for (i in 1:nrow(capture)){

    #This is for the first time you enter to the function
    #It is to get the first channel you started sniffing
    if(channel==0 ){

      channel=capture$Channel[i]

      }

    #A condition to check if the capture has changed of channel and if
      the channel is inside the
    # Channel.Data table to rewrite the inside
    if (channel != capture$Channel[i] && channel %in%
Channel.Data$Channel == TRUE){

      #if it is a new channel you get the time of the previous packet to
            get when it finished sniffing the
      # previous channel
      lastt=capture$Time[i-1]

      #You make an operation to calcute the total time you have been
            sniffing the channel
      time=subset(Channel.Data, Channel == channel)$time.scanning +
(lastt-firstt)

      #NO FUNCIONA SI NO CANVIA DE CANAL EN NINGUN MOMENT (CREC)
      #You get the number of row the current channel is in the
            Channel.Data table
      n=which(Channel.Data$Channel==channel)

      #You create  vector with the time it started and finshed to asniff
                  a channel
      n1<-c(firstt,lastt)
```

```
      #You save the vector in the intervals column which will store all
            the pairs of times
      #This will be useful in other functions
      Channel.Data$intervals[[n]]<- c(Channel.Data$intervals[[n]], n1)

      #You store the moment on time it started to sniff the new channel
      firstt=capture$Time[i]

      #You store the time you calculated previously (This variable
            variates if it founds there has been another
      # period of time analysing the same channel )
      Channel.Data$time.scanning[Channel.Data$Channel == channel ] <-
time

      #Stores the number of the current channel in a provisional variable
      channel = capture$Channel[i]


      #If not, a condition it checks if the current channel in the
            capture does not exist in the Channel.Data table
      } else if( channel != capture$Channel[i] && channel %in%
Channel.Data$Channel == FALSE){

      #Stores the moment of time it stopped analysing the previous
            channel
      lastt=capture$Time[i-1]

      #Gets the total time the previous channel has been sniffed
      time=lastt-firstt

      n1<-c(firstt,lastt)

      #Adds the new channel in the Channel.Data table
      Channel.Data <- rbind(Channel.Data, data.frame(Channel = channel,
time.scanning=time, intervals=NA))

      #Adds the new interval of time in the intervals column
      n=which(Channel.Data$Channel==channel)
      Channel.Data$intervals[n]<- list(n1)

      #Stores the new channel number and the first moment in time of the
                  new channel
      channel = capture$Channel[i]
      firstt=capture$Time[i]

      }

    #Condition to check if you are in the last packet of the capture
    #This is to get the last interval of time of the last channel sniffed
            of the capture
    if (i==nrow(capture)){

      #Condition to check if the current channel is in the Channel.Data
            table
      if (channel %in% Channel.Data$Channel == TRUE){

        lastt=capture$Time[i]
```

```
        time=subset(Channel.Data, Channel == channel)$time.scanning +
(lastt-firstt)

        n=which(Channel.Data$Channel==channel)

        n1<-c(firstt,lastt)

        Channel.Data$intervals[[n]]<- c(Channel.Data$intervals[[n]],
n1)

        Channel.Data$time.scanning[Channel.Data$Channel == channel ]
<- time

        #Condition to check if the current channel is not in the
            Channel.Data table
        }else if(channel %in% Channel.Data$Channel == FALSE){

        lastt=capture$Time[i]

        time=lastt-firstt

        n1<-c(firstt,lastt)

        Channel.Data <- rbind(Channel.Data, data.frame(Channel =
channel, time.scanning=time, intervals=NA))

        n=which(Channel.Data$Channel==channel)

        Channel.Data$intervals[n]<- list(n1)

        }

      }

    }

  return(Channel.Data)

  }
```

```
################################Function 3##############################

#This function will be in charge of getting additional information such
as the retransmitions or the rates
# of the different types of packets and throughput.
extended_aps_table <- function(){

  #Initializing  variables
  totaltime.mf=0.0000
  totaltime.df=0.0000
  totaltime.cf=0.0000
  totalpackets=0.0000
  retransmitted.mf=0.0000
  retransmitted.df=0.0000
  retransmitted=0.0000
  reliability=0.0000
  bytes= 0.0000
  timechannel=0.0000

  #Adding 3 new columns to the aps table
  aps["Management.Rate"] <- 0.0000
  aps["Data.Rate"] <- 0.0000
  aps["Control.Rate"] <- 0.0000
  aps["Throughput.Mbps"] <- 0.0000

  sum=0

  for ( i in 1:nrow(aps)){

    #Creating subset of tables for each type of packets
    provisional.mf <-subset(capture, BSS.Id == aps$BSS.Id[i] &
Type=="Management frame")
    provisional.df <-subset(capture, BSS.Id == aps$BSS.Id[i] &
Type=="Data frame" & !grepl("AD-HOC",capture$DS.status ))
    provisional.cf <-subset(capture, (Transmitter.address ==
aps$BSS.Id[i] | Receiver.address == aps$BSS.Id[i])
                            & Type=="Control frame")


    w=which(Channel.Data$Channel==aps$Current.Channel[i])
    timechannel = Channel.Data$time.scanning[w]


    #Gets the total packets each AP has been transmitting, the number of
            packets that have been retransmitted
    # and calculates the retransmision rate (reliability)
    totalpackets=nrow(provisional.mf)+nrow(provisional.df)
    retransmitted= nrow(subset(provisional.df, grepl("Frame is being
retransmitted",provisional.df$Retry)))
    reliability=retransmitted/totalpackets

    #Vectors with the different channels the AP has been sending
    p.mf <- unique(provisional.mf$Channel, incomparables = FALSE)
    p.df<- unique(provisional.df$Channel, incomparables = FALSE)
    p.cf <- unique(provisional.cf$Channel, incomparables = FALSE)
```

```
    #Loops that go through all the channels
    for ( j in 1:length(p.mf)){


totaltime.mf=totaltime.mf+Channel.Data$time.scanning[Channel.Data$Channel
== p.mf[j]]

      }
    for ( j in 1:length(p.df)){


totaltime.df=totaltime.df+Channel.Data$time.scanning[Channel.Data$Channel
== p.df[j]]

      }
    for ( j in 1:length(p.cf)){


totaltime.cf=totaltime.cf+Channel.Data$time.scanning[Channel.Data$Channel
== p.cf[j]]

      }

    #Calculates the management rate and appends it into the corresponding
AP
    aps$Management.Rate[i] = nrow(provisional.mf)/totaltime.mf

    #Condition to check if there is any Data packets
    if(nrow(provisional.df)!=0){

      aps$Data.Rate[i]=nrow(provisional.df)/totaltime.df
      bytes = sum(provisional.df$Length)

      } else{

        #In case there is not any data packets the data rate will be 0
        aps$Data.Rate[i]=0

      }

    if(nrow(provisional.cf)!=0){

      aps$Control.Rate[i]=nrow(provisional.cf)/totaltime.cf

      } else{

      aps$Control.Rate[i]=0

    }

    #Finally it appends the reliability puts the totaltime to 0 again
    aps$Retransmision[i]=1-reliability

    if(timechannel==0){

      aps$Throughput.Mbps[i]=0
```

```
  }else{

    throughput=(bytes*8)/timechannel
    aps$Throughput.Mbps[i]=throughput/1e6

  }


  bytes=0.0000
  totaltime.mf=0.0000
  totaltime.df=0.0000
  totaltime.cf=0.0000

  }

return(aps)

}
```

```
##########################Function 4################################

#This function will create a table with all the STA detected in the
capture plus additional information
sta_table <- function(){

  #creates a data frame with all the MACs detected
  provisional <- ddply(capture, .(Transmitter.address,Receiver.address),
summarise, RSSI=mean(Signal.strength..dBm.))

  #Converts the vector into a data frame and, at the same time combines
the 2 columns in one
  all.sta<-
as.vector(as.matrix(provisional[,c("Transmitter.address","Receiver.addres
s")]))

  #Discards the duplicated MACs and the broadcast directions
  all.sta<- unique(all.sta,incomparables = FALSE)
  all.sta <- subset(all.sta, all.sta!="" & all.sta!="ff:ff:ff:ff:ff:ff")

  #Vector where it discards all the APs and stores the rest
  all.sta <- all.sta[ which( !(all.sta %in% aps$BSS.Id) )]

  #Converts the matrix into a frame called sta and put a name to the
column
  all.sta1<-matrix(all.sta,nrow = length(all.sta),ncol = 1)
  all.sta1<- as.data.frame(all.sta1)
  all.sta1 <- data.frame(lapply(all.sta1, as.character),
stringsAsFactors=FALSE)
  colnames(all.sta1)=c("Transmitter.address")
  stas <- all.sta1

  #Creates 3 more columns with value 0
  stas$Management.Rate <-0.0000
  stas$Data.Rate<-0.0000
  stas$Control.Rate <- 0.0000

  #Creates 3 provisional variables to get the time it has been receiving
or transmitting each sta
  totaltime.mf=0.0000
  totaltime.df=0.0000
  totaltime.cf=0.0000

  #loop to get through all the STA in the stas table
  for ( i in 1:nrow(stas)){

    #Creates 3 provisional data frames storing the 3 types of packets of
each STA
    provisional.mf <-subset(capture, (Transmitter.address
==stas$Transmitter.address[i] |
Receiver.address==stas$Transmitter.address[i])
                            & Type=="Management frame")
    provisional.df <-subset(capture, (Transmitter.address
==stas$Transmitter.address[i] |
Receiver.address==stas$Transmitter.address[i])
                            & Type=="Data frame" & (grepl("From DS:
0",capture$DS.status )|grepl("To DS: 0",capture$DS.status)))
```

```
    provisional.cf <-subset(capture, (Transmitter.address
==stas$Transmitter.address[i] |
Receiver.address==stas$Transmitter.address[i])
                            & Type=="Control frame")

    #Creates 3 vectors with the channels that have each of the 3 types of
packets
    p.mf <- unique(provisional.mf$Channel, incomparables = FALSE)
    p.df <- unique(provisional.df$Channel, incomparables = FALSE)
    p.cf <- unique(provisional.cf$Channel, incomparables = FALSE)

    for ( j in 1:length(p.mf)){

      #Total time th STA has been transmitting Management frames

totaltime.mf=totaltime.mf+Channel.Data$time.scanning[Channel.Data$Channel
== p.mf[j]]

      }
    for ( j in 1:length(p.df)){

      #Total time th STA has been transmitting Data frames

totaltime.df=totaltime.df+Channel.Data$time.scanning[Channel.Data$Channel
== p.df[j]]

      }
    for ( j in 1:length(p.cf)){

      #Total time th STA has been transmitting Control frames

totaltime.cf=totaltime.cf+Channel.Data$time.scanning[Channel.Data$Channel
== p.cf[j]]

      }

    #Condition to check if there are Management frames
    if(nrow(provisional.mf)!=0){

      #Get the management rate by dividing the number of rows (or
management packets) between the total time
      # stored before
      stas$Management.Rate[i] = nrow(provisional.mf)/totaltime.mf

      } else{

      #In case there are no Management Packets the rate will be 0
      stas$Management.Rate[i]= 0

      }


    if(nrow(provisional.df)!=0){

      stas$Data.Rate[i]= nrow(provisional.df)/totaltime.df

      } else{
```

```
    stas$Data.Rate[i]=0.0000

    }

  if(nrow(provisional.cf)!=0){

    stas$Control.Rate[i]= nrow(provisional.cf)/totaltime.cf

    } else{

    stas$Control.Rate[i]=0.0000

    }

  #Reinicialize the provisional variables
  totaltime.mf=0.0000
  totaltime.df=0.0000
  totaltime.cf=0.0000

  }

return(stas)

}
```

```
##########################Function 5#################################

#This function creates the histogram of all the posible APs
ap_histograms <- function(y){

  #Initialize a list and int variables
  provisional.list <- list()
  j = 0

  #Deletes the folder Histograms and creates a new one
  unlink("Histograms", recursive=TRUE)
  dir.create(file.path("Histograms"))

  #A loop to check all the APs from the aps table
  for (j in 1:nrow(aps)){

    #Creates a subset from the capture table with all the packets related
           with an AP
    approv <- subset(capture, (Transmitter.address==aps$BSS.Id[j]|
Receiver.address==aps$BSS.Id[j])& Channel==aps$Current.Channel[j])

    if (nrow(approv)==0){

      nodata <- data.frame(Time= integer(0), x= numeric(0))
      provisional.list[[paste0("", aps$BSS.Id[j])]] <- nodata

    } else {

      #Transforms the duration of the packets into seconds and orders the
           table by time
      approv1 <- aggregate(approv$Duration/1e6,
by=list(Time=approv$Time%/%1), FUN=sum)

      #Gets the number n of row where the channel the AP is using is in
           the Channel.Data table
      n=which(Channel.Data$Channel==aps$Current.Channel[j])

      #String with the name of the path we want to send the histograms
      folder<- paste("Histograms", aps$Current.Channel[j], sep="/")

      #Condition to check if the folder path exists
      if (!dir.exists(folder)){

        #If it does not exist you create a folder "Histograms"
        dir.create(file.path(folder))

        }

      #z to get the number of times the capture has been sniffing the
           channel
      z=length(Channel.Data$intervals[[n]])


      total=0

      #A loop to that goes from 1 to z jumping 2
      for(i in seq(1,z,2)){
```

```
        #Start to get the moment it started sniffing the channel
        start=Channel.Data$intervals[[n]][i]%/%1

        #Final to get the moment it stopped sniffing the channel
        final=(Channel.Data$intervals[[n]][i+1]%/%1)+1

        #A subset of the approv1 table where the time is bigger than the
            start and lower than the final
        #i.e. a subset of approv1 where the capture was sniffing the
            channel in an interval
        v <- subset(approv1, Time >= start & Time <= final)

        #This is to keep suming the time if there is been diferent
            intervals analysing the same channel
        v$Time= v$Time - start + total



        if (total==0){

          real<- v

          #Add to the provisional list a table with the instant of time
                  and the % of
          # time in the second it has been sending packets of each ap
          provisional.list[[paste0("", aps$BSS.Id[j])]]<- v

          }else {

          #If total!=0 this means the ap table allready exists
          real<- rbind(real, v)

          #Appends the extra information from the current AP
          provisional.list[[paste0("", aps$BSS.Id[j])]]<-
rbind(aps$BSS.Id[j], real)

          }

        #Calculates the total time it is being sniffing a channel
        total=final-start+total

        #It creates a row from the instants of times there is not any
            activity from part of the current AP
        real2 <-
merge(expand.grid(Time=0:(Channel.Data$time.scanning[[n]]+(length(Channel
.Data[[n,3]]))/2)),real, all=TRUE)

        #Changes the NA values for 0. This happenes because you added new
            rows as you can see in the previous line
        real2$x[is.na(real2$x)] <- 0

        #Adds this table into the provisional list
        provisional.list[[paste0("", aps$BSS.Id[j])]]<-real2
```

```
        #Creates a string with the MAC addres of each AP deleting the
            ":"         character
        file.name<-paste0("", aps$BSS.Id[j])
        file.name <- gsub(":", " ", file.name)

        #Creates a png file with the name file.name and stores the
            histogram made from the tables stored in the
        # provisional list
        png(sprintf(paste("Histograms", "%d/%s.png", sep="/"),
aps$Current.Channel[j],file.name))
        if (plothist==TRUE){

          barplot(real2$x, real2$Time, xlab="Time (s)", ylab="time
used in each second (%)", ylim=c(0,yscale),width=1, space = 0.2, col =
"black")

          }else{

            plot(real2$Time, real2$x, type="l", xlab="Time (s)",
ylab="time used in each second (%)",ylim=c(0,yscale), lwd=2,
col="red")

          }

        dev.off()

        }

       }

    }

  #In this loop you will take into consideration the packets that were
      not suposed to be in the channel but are
  # because of the overlapping
  for(i in 1:14){

    #This packets will be called "remains i" where i is the channel
      they  belong to
    name <- paste("remains", i)

    #A subset with all the packets that do not correspond to the
      current channel due to overlapping
    # i.e. packets from APs that should not transmitt in this channel
    approv <- subset(capture, (!Transmitter.address %in% aps$BSS.Id &
!Receiver.address%in%aps$BSS.Id & Channel==i)|(Channel==i &
Current.Channel!=i & !is.na(Current.Channel)))

    if (nrow(approv)==0){

      nodata <- data.frame(Time= integer(0), x= numeric(0))
      provisional.list[[paste0("", name)]] <- nodata

    }else {

      approv1 <- aggregate(approv$Duration/1e6,
by=list(Time=approv$Time%/%1), FUN=sum)


      w<- paste("Histograms", i, sep="/")
```

```
      if (!dir.exists(w)){

        dir.create(file.path(w))

        }

      n=which(Channel.Data$Channel==i)
      z=length(Channel.Data$intervals[[n]])

      total=0

      for(j in seq(1,z,2)){

        start=Channel.Data$intervals[[n]][j]%/%1
        final=(Channel.Data$intervals[[n]][j+1]%/%1)+1

        v <- subset(approv1, Time >=start & Time<=final)

        v$Time= v$Time - start + total

        if (total==0){

          real<- v
          provisional.list[[paste0("", name)]]<-v


          }else {

          real<- rbind(real, v)
          provisional.list[[paste0("", name)]]<-rbind(name, real)

          }


        total=final-start+total

        real2 <-
merge(expand.grid(Time=0:(Channel.Data$time.scanning[[n]]+(length(Chan
nel.Data[[n,3]]))/2)),real, all=TRUE)
        real2$x[is.na(real2$x)] <- 0

        provisional.list[[paste0("", name)]]<-real2

        png(sprintf(paste("Histograms", "%d/%s.png", sep="/"),
i,name))

        if (plothist==TRUE){

          barplot(real2$x, real2$Time, xlab="Time (s)", ylab="time
used in each second (%)",ylim=c(0,yscale),width=1, space = 0.2, col =
"black")

          }else{

            plot(real2$Time, real2$x, type="l", xlab="Time (s)",
ylab="time used in each second (%)",ylim=c(0,yscale), lwd=2,
col="red")

            }
        dev.off()
```

```
        }
      }
    }
  return(provisional.list)
  }
```

```
############################Function 6#################################

#Gets the total correlation between each pair of APs in the same channel
total_correlation <- function(i){

  #Subset with all the APs that work in the same channel
  provisional <- subset(aps,aps$Current.Channel==i )

  #Creates a data frame with only the names of the APs + the "remains i"
      in each row
  aps.name <- data.frame(provisional$BSS.Id, stringsAsFactors=FALSE)
  aps.name <- rbind(aps.name, paste0("remains ", i))


  #A loop that goes through all the aps in the channel
  for(j in 1:nrow(aps.name)){

    #Creates a column with each name of AP
    aps.name[paste0(aps.name[j,1],"")]<-NA

    }

  for (j in 1:nrow(aps.name)){

    for (l in 1:nrow(aps.name)){

      if(j!=l){

        nrow1 <- sapply(apinfo[paste(aps.name[j,1],"",sep="")],
nrow)[[1]]
        nrow2 <- sapply(apinfo[paste(aps.name[l,1],"",sep="")],
nrow)[[1]]

        if (nrow1!=0 && nrow2!=0){

          #Calculates the correlation between each pair of APs
          correlation <-
cor(as.data.frame(apinfo[paste(aps.name[j,1],"",sep="")])[2],

as.data.frame(apinfo[paste(aps.name[l,1],"",sep="")])[2])

          #Adds the obtained correlation to the aps.name table
          aps.name[j,l+1]<-correlation[1,1]

        } else{
          aps.name[j,l+1]<-0.0000


            }
        }

      }

    }

  return(aps.name)
}
```

```
##############################Function 7#################################

#Gets the correlation each x seconds between each pair of APs in the same
channel
correlation_xseg <- function(i,x){

  if (i %in% Channel.Data$Channel){

      provisional <- subset(aps,aps$Current.Channel==i )
      provisional <- rbind(provisional, paste0("remains ", i))

      #Stores the the total time it has been sniffing the channel into
            the time variable
      time <- (Channel.Data$time.scanning[Channel.Data$Channel==i]%/%1)+1

      #Stores the time divided by x
      tentime <- (time%/%x)+1

      #Creates a table with the times jumping x units every time as rows
      provisional2 <- expand.grid(Time=seq(x,tentime*x,x))
      k <- 1

      apsname <- provisional$BSS.Id

      #Condition to check if there is 2 or more APs in the channel
      if(length(apsname) >= 2){

        for(j in 1:(length(apsname)-1)){

          if(j!=length(apsname)){

            h <- j+1

            }

          for (l in h:length(apsname)){

            #k to know in what x seconds we are calculating the
                  correlation
            k<- k+1

            #Creates a column with the name of the pair of APs
            provisional2[paste(apsname[j],apsname[l],sep = " // ")]<-NA

            for (r in 1:nrow(provisional2)){

              if(r!=nrow(provisional2)){

                #gets the x seconds info of the first ap
                interval1 <-
as.data.frame(apinfo[paste(apsname[j],"",sep="")])[(provisional2[r,1]-(x-
1)):provisional2[r,1],2]

                #gets the x seconds info of the second ap
                interval2 <-
as.data.frame(apinfo[paste(apsname[l],"",sep="")])[(provisional2[r,1]-(x-
1)):provisional2[r,1],2]
```

```
                    if(any(is.na(interval1))==FALSE){
                      sdinterval1<- sd(interval1)
                    }else{
                      sdinterval1 <- 0
                    }
                    if(any(is.na(interval2))==FALSE){
                      sdinterval2<- sd(interval2)
                    }else{
                      sdinterval2 <- 0
                    }

                    if (sdinterval1!=0 && sdinterval2!=0){

                      correlation <- cor(interval1,interval2)

                      provisional2[r,k]<- correlation

                    } else {

                      provisional2[r,k]<- 0.0000

                      }

                    }else if (r == nrow(provisional2)){ # last x seconds
                            might not be x seconds

                    #Gets the last seconds sniffed of the first AP
                    interval1 <-
as.data.frame(apinfo[paste(apsname[j],"",sep="")])[(provisional2[r,1]-
(x-1)):time,2]

                    #Gets the last seconds sniffed of the second AP
                    interval2 <-
as.data.frame(apinfo[paste(apsname[l],"",sep="")])[(provisional2[r,1]-
(x-1)):time,2]

                    if(any(is.na(interval1))==FALSE &&
length(interval1)>1){
                      sdinterval1<- sd(interval1)
                    }else{
                      sdinterval1 <- 0
                    }
                    if(any(is.na(interval2))==FALSE &&
length(interval2)>1){
                      sdinterval2<- sd(interval2)
                    }else{
                      sdinterval2 <- 0
                    }

                    if (sdinterval1!=0 && sdinterval2!=0){

                      correlation <- cor(interval1,interval2)
                      provisional2[r,k]<- correlation

                    }else {

                      provisional2[r,k]<- 0.0000

                      }
```

```
                }
            }
        }
    }
}
return(provisional2)
  }
}
```

```
#########################Function 8#################################
Relation <- function(i, limitcor){

# Checks What APs are related between them
if(exists(paste0("segCor",i))){

    cor<- get(paste0("segCor",i))

    #Get the mean of the correlation
    cor<-colMeans(x=cor, na.rm = TRUE)
    cor <- t(cor)

    #Removes the Time column
    cor<-subset(cor, select= -c(Time))


    if( ncol(cor)>0){

      #Adds 2 cells in each column with the AP name of each column
           title
      cor <- rbind(cor, lapply(strsplit(colnames(cor), " // "),
"[",1))
      cor <- rbind(cor, lapply(strsplit(colnames(cor), " // "),
"[",2))

      #Creates a table with the APs name of each channel and a
           Relations column of type list()
      x <- data.frame(AP= character(nrow(subset(aps, Current.Channel
== i))))
      y <- subset(aps, Current.Channel == i)
      x$AP <- y$BSS.Id
      x$Relations <- list(0)

      #Discards all the relations bigger than the limitcor global
           variable
      cor<-cor[,cor[1,]< limitcor, drop=FALSE]

      m<- list()

      if(ncol(cor)>0){

        for (j in 1:ncol(cor)){

          if (!grepl("remains", cor[3,j] ) ){

            #Checks the position of the Aps in the x Table
            n= which(x$AP == cor[2,j] | x$AP == cor[3,j])

            #Vector with the 2 APs related
            m= cor[2:3, j]

            #Adds the names of the AP related with it
            x$Relations[[n[1]]] <-c(x$Relations[[n[1]]], m[!m %in%
x$AP[n[1]]])

            x$Relations[[n[2]]] <-c(x$Relations[[n[2]]], m[!m %in%
x$AP[n[2]]])

            x$Relations[[n[1]]] <- unique(x$Relations[[n[1]]])
            x$Relations[[n[2]]] <- unique(x$Relations[[n[2]]])
```

```
            }
          }
        }
        return (as.data.frame(x))
        }
      }
}
```

```
#####################Global variables to configure#####################

#Set the scale of the y axis in the histograms
yscale <-0.4

#Set path where the file with all the capture from wireshark is
path <- "path"

#Sets the the interval of the correlations between 2 APs (has to be
bigger than 1)
cortime<-5

#limitcor
limitcor=-0.3

#type TRUE if you want the plots as histograms or FALSE if you want a
line in time
plothist= TRUE

#########################Calling functions##########################

#first

data <- export_aps_data(path)
capture <- data$capture
aps <- data$aps

#RSSI

rssi()

#second

Channel.Data <- channel_data(capture)


#third

aps <-extended_aps_table()

#forth

sta<- sta_table()

#fifth

apinfo<-ap_histograms(yscale)

#sixth

for (i in 1:14){

  assign(paste0("Cor", i), total_correlation(i))

}
```

```r
#seventh

for (i in 1:14){

  x <- correlation_xseg(i, cortime)

  if(!is.null(x) && ncol(x) > 1){

  assign(paste0("segCor", i), x)

  }
  rm(x)

}


#eighth

for (i in 1:14){

  y <- Relation(i, limitcor)

  if(!is.null(y)){

    assign(paste0("Relations", i), y)

  }
  rm(y)

}
```