Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Telecommunication Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

# Counting people through Wi-Fi probe detection on a low-cost sensor

Relatore: Prof. Alessandro Enrico Cesare Redondi
Correlatore: Prof. Matteo Cesana

Tesi di laurea di:
Paolo Eugenio Galluzzi Matr. 835876

Anno Accademico 2016–2017

# Abstract

What if you could know the study hall's occupancy before leaving your house? What if, during a hard study period, you could know what classrooms are completely free? It would be useful and could sometimes saves you some precious time, right?

The main goal of this thesis project is to develop a low-cost occupancy sensor, that can estimate the number of people in classrooms, laboratories, study halls and other environments frequented by the students of Politecnico di Milano. The estimation is carried out through the Wi-Fi probes detection and count. These probes are particular types of Wi-Fi packets, which contain the unique identification number of the sender devices.
Along with the occupancy estimation, the sensor provides also environmental data like temperature, humidity, pressure and brightness. All the data collected are uploaded to a public website, to make them easy to reach by students. Along with the typical website method, a chat-bot is implemented, in order to make the data consulting quicker by students using smartphones, without loading the entire website. We think that this last solution will be more appreciated by those students who frequently use messaging apps, rather than websites, to retrieve informations.
We achieved promising results, that makes us better comprehend the potentiality of this non intrusive Wi-Fi count method. If combined with environmental data, we believe that this technique can lead to further steps, in order to achieve the higher goal of a smart campus development.

**Keywords:** IoT, Smart Campus, Wi-Fi probe detection, Wi-Fi sensing, Indoor Occupancy count.

# Sommario

E se potessi sapere il grado di occupazione della biblioteca prima di uscire di casa? E se, durante un forte periodo di studio, potessi sapere quali aule sono completamente libere? Sarebbe utile e ti potrebbe far risparmiare del tempo prezioso, giusto?

Lo scopo principale di questo lavoro di tesi è quello di sviluppare un sensore a basso costo, capace di stimare il numero di persone presenti nelle classi, nei laboratori, nelle aule studio e in altri ambienti frequentati dagli studenti del Politecnico di Milano. La stima viene effettuata attraverso la cattura e il conteggio dei probe request del protocollo Wi-Fi. Questi probe sono dei particolari tipi di pacchetti Wi-Fi, che contengono l'identificativo univoco del dispositivo che gli invia.
Assieme alla stima dell'occupazione dell'aula, il sensore fornisce anche misure sui dati ambientali quali temperatura, umidità, pressione e luminosità. Tutti questi dati sono poi caricati su un sito di pubblico dominio, di modo che gli studenti possano consultarli facilmente. Oltre al classico sito internet, viene fornito anche un chat-bot, di modo che i dati siano più velocemente consultabili dagli studenti muniti di smartphone, senza aver bisogno di caricare l'intero sito. Pensiamo che quest'ultima soluzione verrà parecchio apprezzata dagli studenti che usano spesso applicazioni di messaggistica, rispetto ai siti, per ricevere informazioni.
Abbiamo ottenuto risultati promettenti, che ci hanno fatto capire meglio le potenzialità di questo metodo di conteggio non invasivo. Se combinato con i dati ambientali crediamo che questa tecnica potrà portare a grossi passi avanti, con l'obiettivo di raggiungere presto lo sviluppo di uno Smart Campus.

**Parole Chiave:** IoT, Smart Campus, Cattura dei probe Wi-Fi, Wi-Fi sensing, Conteggio dell'occupazione in interni.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Overview

Since 2007 there has been a rapid rise in the use of Wi-Fi enabled devices. Nowadays the majority of people uses more than one of these devices, like smartphones, tablets or notebooks. Some journalists defines nowadays smartphones as: "the remote-controls of our lives". Sure enough what in the past were notebooks are today's smartphones and tablets: they contain everything, from our social contacts to our work stuff.
According to some interesting statistics of 2014:

1. There will be more than 7 billion new Wi-Fi enabled devices in the next 3 years. – Sys-Con

2. Wi-Fi has become the medium of choice. 2/3 of US consumers prefer Wi-Fi to Cellular. – Deloitte

3. By 2017, 60% of carrier network traffic will be offloaded to Wi-Fi. – Wireless Broadband Association

4. 79% of smartphone users have their phone with them for 22 hours a day. – IDC

5. 67% of smartphone users check for messages without being notified. – Pew Research Center

Considering that we can reasonably assumes this statistics valid also nowadays, it seems that without Wi-Fi or some sort of wireless connection to internet, we are completely lost.
If we consider that streaming platforms (e.g. Netflix, Google Play Films, Spotify etc.)

are in continuous growth[1], and that these contents are mostly enjoyed using mobile devices, we can see the importance of Wi-Fi.
Let's now focus on a university campus: this place is a melting pot for students of different parts of the world. For some of them, Wi-Fi is the only connection they use inside the campus. Students can therefore benefit of the university connection in almost all the campus' places. Although here is the interesting thing: when the Wi-Fi of our laptops, smarphones or tablets is on, our device sends packet of different types. Some of them can require a connection to a website, or bring voip traffic, or multimedia content. Typically, when connected to an AP, these packets are encrypted, in order to protect your privacy during web surfing. However some of these packets are not, and they are particular packets called "probes". When your device is searching for available networks, it sends specific packets called "Probe Requests". These packets are a sort of "Passport" of your device, that tells the device's name (the name of the manufacturer of the device and the unique number of your device) and, in some cases, also where you have been (the names of other Wi-Fi access points that you've saved [4]).

## 1.2 Project Objective

The main purpose of this project is to lay the foundations of a system that can improve students' experience at the university campus. In the era of the Internet of Things, this is a way of moving towards the concept of "smart campus".
Let's clarify these statements with an example: suppose that you're a student in a hard exam session, and you need to choose between two study halls. They are pretty far apart thus you want to make sure that, when you get there, there is at least one free spot where you can study. Suppose also that there are others free spaces e.g. classrooms, and you want to find the one with enough free space for you and your colleagues. It would be easier and faster to know this information without having to check yourself all the places, right? So why don't deploy, in every room of the campus, a sensor which can collect and provide these data?
Until the past years make this sort of measurements requires high cost. A computer with a Wi-Fi card able to capture nearby traffic must be used, and therefore the costs and space requirements were higher. Nowadays cheap electronic sensors can be found at low price, around 2€ or less, and microcontrollers becomes more powerful, cheapest and simpler to use. For these reason, the goal of this thesis project is to develop a Wi-Fi

[1]Media Consulting estimates that the streaming platforms will cover over the 20% of the "pay" market by 2018

sensor that can estimate people's presence through probe request capture, along with measuring environments data like temperature, humidity, pressure and brightness. All the data collected must be anonymised and sent to a central server that makes these data available to all the students.

## 1.3 Thesis Outline

This section summarizes the layout and contents of the thesis' chapters.

**Chapter 1** presents the context in which the thesis is carried out and defines the goals of this research. The main goal is related to the development of a device capable of detecting and estimating the number of people in a room. Along with this estimation, the device provides some ambient measurements, that could be useful to the students and for future analysis. This chapter also provides the summary of the thesis and outlines the contents of this work.

**Chapter 2** provides an overview of the Wi-Fi sensing and probe request analysis panorama. It describes the state of the art for indoor and outdoor occupancy detection and tracking, with the use of Wi-Fi and other environment data. Moreover there is an analysis of the random MAC phenomena, with a hint on the privacy issues relating to these analysis.

**Chapter 3** introduces the theoretical basis of this work. It explain in detail how Wi-Fi works, with a particular focus on probe requests. Furthermore the protocol used for the device-to-server communication (MQTT), and the program used to manage and display all the informations to the users are presented.

**Chapter 4** describes the system's implementation part. It's divided in two main parts: a hardware part, and a software part. The first one describes how the sensors works and how they are connected to the main microcontroller. It explain also how, from a first prototype, we create a final Printed Circuit Board, smaller and more practical than the first one. In the final section of this first part, we explain the program's functions through a flowchart, and introduce the estimation's model and its characteristics.
The second part explains the software implementation: the server set-up, the communication server configuration, the displaying of the outputs, and the possible methods for data visualization at users' disposition ( the website and chat-bot).

**Chapter 5** its the model's performance test part. Here we explain our dataset and validation model used. Then we show our model's performance in terms of error between the estimation, and the real number of people. The performances are evaluated for every dataset in our possession, and followed by some graphs that helps the results' explanation. At the end of the chapter a summary table is presented and commented, in order to give some explanations to the final results.

**Chapter 6** is the final conclusive part of the project. Our initial goals and their achievement are discussed, leaving then the space for some considerations about possible future works.

# Chapter 2

# State of the art

This chapter gives an overview of the current advancements in indoor and outdoor occupancy detection using Wi-Fi. The main goal is to provide a wide view on the state of the art for the subject of this work. This analysis serves as a common base for all the investigation, decisions and results presented later on.
The first section is an analysis on how talkative are our devices, thanks to the data presented on [12] by Julien Freudiger.
In the second section we discuss the feasibility of using these captured data to actually estimate the real number of people, in an indoor and outdoor environment. Among the plenty of papers regarding this subject, we chose the ones that seems to be more suitable for our purposes, like [11], [19], [7] and [15]. In this section we also mention a peculiar use of probe request's capture, thanks to the work of V. Acuna et al [1].
In the third section we introduce the random MAC problematic, and some possible solutions.
At last in the fourth section we analyse the use of indoor measurements such as light, temperature, humidity and $CO_2$ in order to help the Wi-Fi occupancy estimation. For this purpose we present the work of Luis M. Candanedo et al. [8], where the occupancy detection is obtained only with light, temperature, humidity and $CO_2$ measurements.

## 2.1 Wi-Fi Probe frequency

One of the challenges of wireless networks is their discovery process. The IEEE 802.11 standard defines two mechanism of announcing the device's presence: the active scanning, and the passive scanning. While the passive scanning will be explain later and it's not important for our purposes, the active scanning is the one we're really interested about. Mobile devices can proactively discover APs by sending probe re-

quest frames (fig. 3.6 of then next chapter) on the various Wi-Fi channels. Since IEEE standard doesn't impose a broadcasting frequency for probe requests, the scheme used by devices depends on their Operating System. By actively scanning the channels, devices can keep the Wi-Fi radio on for just few milliseconds.
Probe requests are practically a unique fingerprint since they contain the MAC address of the mobile devices. Users are mostly unaware of the fact that their devices broadcast probe requests. For this reason Julien Freudiger in [12] analyse the probe request's frequency of various devices and OSs.
The measurements are initially divided in six hardware configuration, in order to understand which configuration captures the largest number of probes.

1. **dynamic** where there is one antenna hopping across 802.11b/g channels.

1. **static** where the antenna is fixed at one of the three non overlapping channels of 802.11b/g

3. **dynamic** where there are three antennas hopping accross 802.11b/g channels in a coordinated fashion

3. **dynamic.s** where the three antennas haven't coordinated hopping

3. **dynamic.s.n** where the three antennas explore also the n fashion (5GHz)

3. **static** where the three antennas are fixed one in every non-overlapping 802.11b/g channel (1,6,11).

After these tests, the configuration that gives the better output was the last one, with the three antennas fixed on the three non overlapping channels of Wi-Fi b/g. Once selected the hardware configuration, the test involves four different types of devices: a Samsung with Android 4.4, LG with Android 5.0.1, Iphone 6 with iOS 8.1.3 and a Blackberry with its proprietary OS. These devices were tested in different configuration: with a different number of APs saved (until 20 APs), while on charging, with screen on, Wi-Fi settings opened and so on.
In conclusion this work shows that probe requests are bursty in nature and their behaviour depends on the OS version. Android devices are particularly "talkative" (approximately a burst of probes every minute), and their number of probes (in case of Android 4.4) increase linearly with the number of SSIDs in memory. Approximately we're talking about a number of probes between 2000 and 3000 per hour on Android devices, less than 200 for iOS and 0 for Blackberry devices.
In conclusion we can say that Wi-Fi probes offers a good parameter for presence estimation in public areas.

## 2.2 Estimating occupancy through probe request

We have seen that Wi-Fi probe requests are a big privacy threat, and that different devices produce a different amount of this packets. In [11] this fingerprints are used in order to estimate transit passenger population at a bus stops.
This work presents the sequent question: how it is possible to improve the quality of public transportation? The work proposes a solution based on an estimation of the passenger population at bus stops, in order to optimize bus routes in real-time and increase the quality of service. Two stops are taken as reference: the first one in an uncrowded suburban environment, and the second one in a crowded terminus one. The hardware of the system is base on a Raspberry Pi (a single-board Linux computer using an ARM-based system-on-a-chip) with a USB Wi-Fi adapter capable of monitor mode. This hardware costs almost 30€, while the one we used in our project is not a computer but only a microcontroller, and costs about 2€.
The system uses channel hopping during monitor mode, in order to scan more channel. With Airodump-ng (a software for sniffing Wi-Fi packets) a log file is created, from which their model applies the next filters:

**RSSI** - is used for filtering according to the distance from the bus stop. Because signal strength depends on the distance between the receiver and the device, if the device is further away, the signal strength is weaker.

**Number of received packets** - if the number of received packets is less than 2 they are discard because supposed to belong to passers by.

**Time** - they use two limits: less than one minute duration indicates a "too-short" time, while more than thirty minutes a "too-long" time. Devices belonging to the first case probably represent people passing by, while devices from the second case represent permanent devices in residential or office environments in the proximity of the bus stop.

After applying these filters, the data collected shows a lower error rate in the crowded environment rather that in the suburban, less crowded environment. The error was calculated as:

$$e_i = \frac{a_i - o_i}{o_i} 100 \tag{2.1}$$

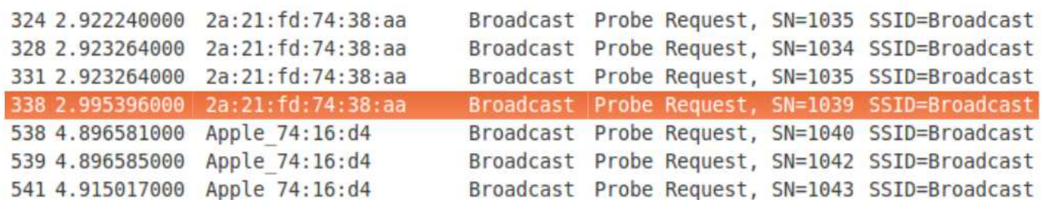
where $a_i$ and $o_i$ are the approximate number and exact number of passengers respectively. In the first environment they obtain a maximum error of 13.71%, while in the suburban environment the maximum error reaches 30.91%.
In conclusion the work of Thongtat Oransirikul et al. verified a useful relationship

between distance and RSSI values: this last one decreases proportionally with distance and independently on the Wi-Fi device type. By delaying their estimation by one minute they were also able to obtain a near real-time estimation of passenger numbers, with a moderately-high positive correlation to the observed numbers.
Although they used a more powerful device (the Raspberry Pi), and makes all the analysis offline, the work of this team is similar to the one we want to implement. However, we want to use a low cost chip without too much computing capabilities. Also our goal and dataset is different, and composed by students having classes or studying in indoor environments.
Related to the indoor occupancy detection using Wi-Fi, the work of Edwin Vattapparamban et all. [19] shows that, using probe requests, it is possible to track the occupancy in different zones inside a building. This team have used a hardware called "WiFi Pineapple", a device with dual integrated radios and Atheros AR9331 System on a Chip, with 16MB ROM and 64MB of RAM. All these devices are accessed using their web interface built on a Unix machine called BusyBox. The software used to capture WiFi probe requests is tcpdump, with a filter to reject packets with an RSSI value less than -100dBm or higher than -30dBm. The collected MAC addresses were hashed, in order to preserve anonymity of the users. Since the final goal of this work was the real-time occupancy monitoring, an algorithm was used for tracking the users and take advantage of their past location estimations. The WiFi-PA were positioned in a particular "grid", in order to map the different RSSI values with a precise position on the grid. To get a more accurate location information and triangulate users, their RSS informations had to be captured by at least 4 WiFi-PAs. Other than the fact that in our case we're not interested in users exact location, this work uses, like the previous, a more powerful SoC with more than one WiFi modem. In conclusion to [19], they achieved to estimate the location of a mobile device based on probes information captured at multiple reference locations. This results shows that probe request can, again, be a valuable solution for occupancy monitoring in future smart buildings.

A final particular mention is dedicated to the work of V. Acuna et al. [1]. In this work the people's localization through Wi-Fi probe capture is applied in important application such as search and rescue operations. In this development, a WiFi-PA Mark V (like the one used in [19], useful since can be battery-powered) is used attached to a UAV (Unmanned Aerial Vehicle) to estimate user's location in case, for example, of a natural disaster. The WiFi-PA is mounted on the UAV along with a smartphone, which provides internet connectivity to the WiFi-PA and therefore its remote control. Moreover, the random forest machine learning algorithm is used to localize the Wi-Fi devices into predefined zones, based on the observation of the RSSIs from Wi-Fi devices. In conclusion the results of the project shows an accuracy of 81.8% for finding the true zone of a Wi-Fi device.

## 2.3 MAC Address Randomization

In order to protect users' privacy, since version 8 of iOS, Android 6.0, Windows 10 and Linux kernel 3.18, the MAC address randomization measure was adopted. The exact methods, timing and MAC generated depends on the OS, but in line of principle this technique allows the device to send probe request with different MAC addresses that the real one they possessed.
In [9] a timing-based attack is presented, in order to defeat MAC randomization techniques and discover the real MAC of the device. This attack considers the inter-frame arrival time between frames using the same MAC address. Evaluating the distances, the algorithm estimate if two or more groups of frames with different MACs comes from the same device or from different ones.
Although this method is promising, for our purpose we only want to divide random MACs from non random ones. From our results, and those of [12] and [19], we founds that it's easy to detect random MAC addresses. Organizations developing products using Wi-Fi must register to IEEE MAC Address list [5]. Unless a device generates a random MAC address with an assigned OUI identification, random MAC addresses are easy to recognize since they'll not correspond to any of the manufacturer on the OUI list. Software like Wireshark automatically detects MAC address included in the OUI list, and label them with the manufacturer's name.
A more careful analysis, brought up by Julien Freudiger, consist in an analysis of the sequence numbers. This progressive numbers shows that it's possible to link different packets with different MAC addresses, sent by the same device. As we can see in figure 2.1, device 5a:e3:24:ea:35:4a broadcasts a probe request with SEQ=1039 and just after this event, a device Apple_51:2d:db broadcasts a probe with SEQ=1040. It's



```
324 2.922240000  2a:21:fd:74:38:aa      Broadcast  Probe Request, SN=1035 SSID=Broadcast
328 2.923264000  2a:21:fd:74:38:aa      Broadcast  Probe Request, SN=1034 SSID=Broadcast
331 2.923264000  2a:21:fd:74:38:aa      Broadcast  Probe Request, SN=1035 SSID=Broadcast
338 2.995396000  2a:21:fd:74:38:aa      Broadcast  Probe Request, SN=1039 SSID=Broadcast
538 4.896581000  Apple_74:16:d4         Broadcast  Probe Request, SN=1040 SSID=Broadcast
539 4.896585000  Apple_74:16:d4         Broadcast  Probe Request, SN=1042 SSID=Broadcast
541 4.915017000  Apple_74:16:d4         Broadcast  Probe Request, SN=1043 SSID=Broadcast
```

*Figure 2.1: A Wireshark screen-capture - by Julien Freudiger*

reasonable to assume that the incremental SEQ indicates that both packets might be

originated from the same Apple device. In the same figure we can also see what we've said before, that if the MAC address is not registered into the IEEE MAC Address, Wireshark doesn't recognize it as a registered manufacturer.
For our purposes this kind of detection of ID's randomization is enough to create two groups of MAC Addresses: the random and the non-random ones.

## 2.4 Occupancy detection through ambient measurements

In this last part of the chapter we want to introduce a study on accurate occupancy detection using only data of temperature, brightness, humidity and $CO_2$ [8]. Other works in this direction ([2], [3]) reports that, thanks to occupancy data as an input for HVAC[1] control algorithms, energy savings can raise between 29% and 80% inside buildings. The work [8] exploits environment data using a Raspberry Pi, a microcontroller connected to a DTH22 low-cost temperature and humidity sensor, a camera, light sensor and $CO_2$ sensor. The data is collected within an office of 5.85m x 3.50m x 3.53m (W x D x H), and the measurements, along with the camera time stamps, are sent via ZigBee standard. The obtained datasets are then used to train and test four classification models: CART, RF, GMB and LDA.
The Classification And Regression Trees models stratify the region where the predictions are done, into a number of simple regions.
Random Forest is instead a model that makes an effort to improve the accuracy of prediction by creating many classification trees. During the training, a set of observations not used to obtain the trees, are used to estimate the error and are therefore referred as "the out-of-bag" observations.
The GBM model tries to improve the prediction for a decision tree by using information from previously generated trees.
LDA model uses Bayes theorem to estimate probabilities under the assumption that each of the variables follows a normal distribution.
In conclusion, the work shown that it's possible to obtain high accuracy in the determination of occupancy by the sole use of ambient parameters. High accuracy (around 97%) were found when using only two predictors e.g. temperature and light, with LDA models.
This is an important achievement, that could be fundamental in developing and support a more precise estimating sensor.

[1]Acronym that stands for Heating, Ventilation and Air Conditioning.

# Chapter 3

# Theoretical considerations

This chapter is intended as a review of the theoretical fundamentals used in this thesis. The aim is to better comprehend the working principles of the technologies used in this work.
In order to do this, the first part of this chapter is a review of the IEEE 802.11 standard: its amendments, the MAC frame, its functions and in particular the discovery function. In the second part the MQTT protocol for IoT is introduced, along with the NODE-RED programming tool. The third and last part is an overview of the microcontroller panorama, with particular focus on the ESP8266 family.

## 3.1 The IEEE 802.11 Standard

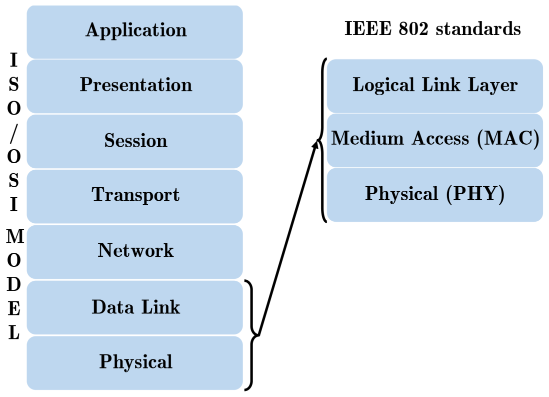


*Figure 3.1: OSI model and IEEE 802.11 interested layers*

In computer science the IEEE 802.11 standard, better known as Wi-Fi, is a set of Medium Access Control (MAC) and Physical (PHY) recommendations for implementing a WLAN (Wireless Local Area Network) communication between devices. The

standard is created and maintained by the Institute of Electrical and Electronic Engineers (IEEE) and its first version was created in 1997. Since then the standard has evolved to introduce more functions and capabilities (such as QoS in the 802.11e standard) either at MAC and Physical layer. The layer of the OSI Model[1] in which IEEE 802.11 standard operates is shown in figure 3.1.
Some of the most relevant component of a Wi-Fi network (see Figure 3.2) are:

- Station (STA): the wireless terminal (Computer, smartphone, sensor...)
- Access Point (AP): it's the "bridge" between wireless and wired network
- Basic Service Set (BSS): set of terminals regulated by the same coordination function [2]. It can be related to the concept of "cell" in a mobile radio network
- Extended Service Set (ESS): set of infrastructure BSS [3].

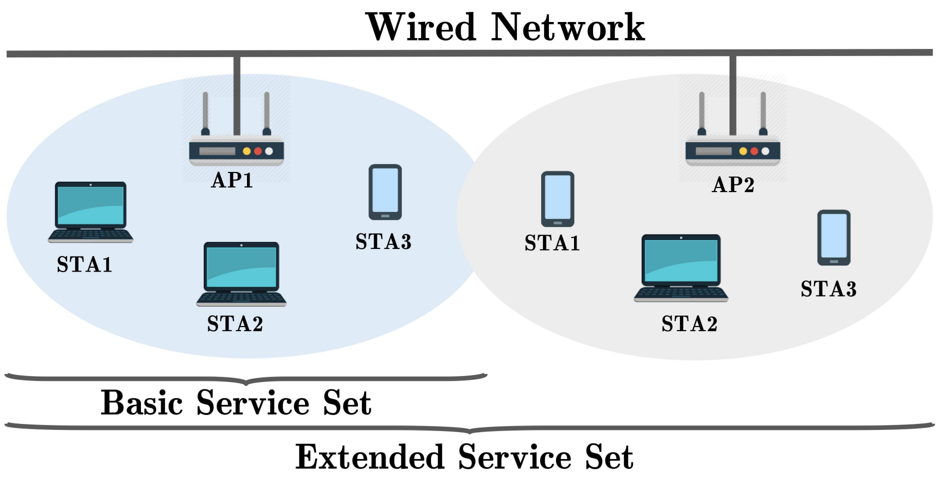


*Figure 3.2: 802.11 Architecture*

Without entering in too much details, in the sequent sections the physical and MAC layers of the 802.11 standard are described.

### 3.1.1 Physical Layer

The legacy version of 802.11 defines three techniques to transmit data and reduce interference: Infrared (IR), Frequency Hopping Spread Spectrum (FHSS) and Direct

[1]The Open System Interconnection model was the first model to introduce the concept of protocol layer architecture, with 7 layers. Its aim was to standardize computer networking.

[2]The c.f. is a logic function that manages the access to the radio channel e.g. decides whether station A or station B transmits data

[3]In the infrastructure BSS all communications have to be with the AP. Direct connections between terminals are not possible, and have to use the access point's relay function.

Sequence Spread Spectrum (DSSS).
The infrared technology is based on the 850-950nm wavelength, and this lead to some disadvantages like the limited coverage radius and the impossibility to overcome obstacles like walls. For these reason, and for the slow 1MBit/s rate, this technique is obsolete and no longer used.
Frequency hopping spread spectrum is a technique based on the artificial expansion of the band of the signal. It guarantees better performance against noise [4] . In the FHSS case, the spreading is obtained dividing the band in sub-bands of 1MHz, in order for the transmitter to change the sub-channel in a given sequence. To avoid interference, near stations must have orthogonal hopping sequences. This technique guarantees a rate between 1 and 2 MBit/s but due to these poor performance is no longer used in Wi-Fi.
Direct sequence spread spectrum, as FHSS, is based on the expansion of the signal band. In this case the expansion is obtained by means of the multiplication of the signal by a sequence called "barker sequence"[5] of higher rate. The bandwidth is proportional to the rate of the transmitted signal, therefore higher rate (like that of the barker sequence) coincide to larger bandwidth.

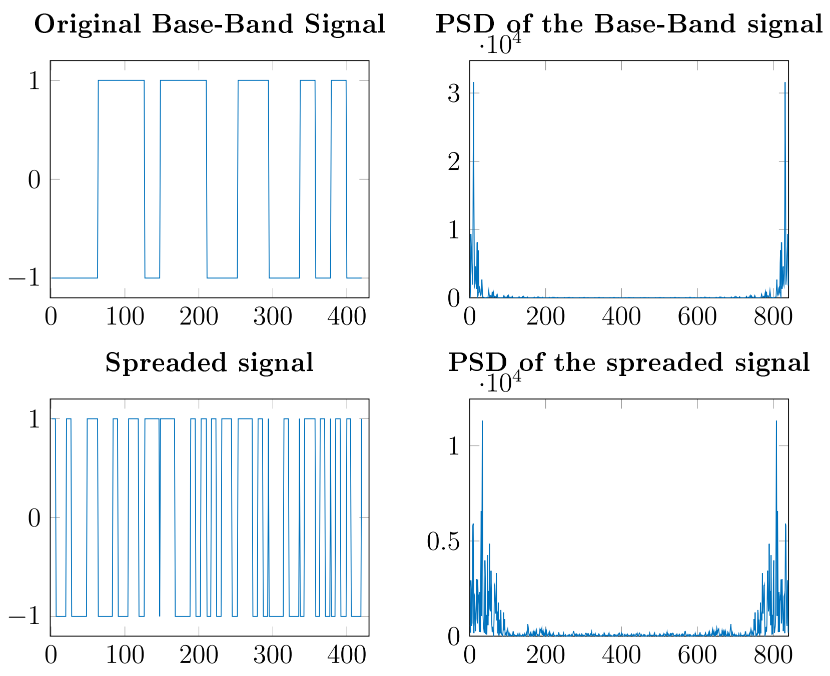


*Figure 3.3: DSSS example*

[4] «The term *noise* is used customarily to designate unwanted waves that tend to disturb the transmission and processing of signals in communication systems and over which we have incomplete control»[17, p. 179]

[5] Example of a data stream and random barker sequence multiplication can be found in the Appendix A

The power of the signal is therefore distributed along the spectrum and, at the receiver, this technique improve the signal to noise ratio [6] thus the performance and robustness of the system. The figure 3.3 shows the Power Spectral Density (PSD) in relation to two different signals: the "base-band signal" has lower rate and thus lower bandwidth requirements; the "Spreaded Signal" is the original signal multiplied by the barker sequence with higher rate; this causes an enlargement of the occupied bandwidth, as seen in the right corner figure. Since in DSSS the multiplication with a higher rate code has only a spreading purpose (in contrast with the CDMA technique in which has also a multiple access purpose), all the transmissions inside the same BSS use the same barker sequence. DSSS guarantees a bit rate between 1 and 2 MBit/s.
The DSSS technique operates in the ISM band [7] among 2,4 and 2,4835 GHz, with a total range of 83,5 MHz. Within this interval 14 channels are defined, with bandwidth of 22 MHz.
Table 3.1 shows the availability and frequencies of the Wi-Fi channels in various regions. In U.S.A. and Europe the three non overlapping channels are 1, 6 and 11. In terms of modulation, the DSSS uses DBPSK and DQPSK.

| Country | Available Channels |
|---|---|
| U.S.A. | 1-11 (2,412-2,462 GHz) |
| Europe | 1-11 (2,412-2,472 GHz) |
| Spain | 10-11 (2,457-2,462 GHz) |
| France | 10-13 (2,457-2,472 GHz) |
| Japan | 14 (2,484 GHz) |

*Table 3.1: Channels Availability*

The performance, in terms of bit rate, of the above-mentioned techniques, are really low. Due to this and other factors (such as frequency conflicts or QoS[8] requirements), during the years some amendment to the legacy version of 802.11 were done, characterized with a letter beside the number of the standard:

**802.11b** Introduced in 1999 it uses a spread spectrum technique known as CCK (Complementary Code Keying). This technique can adapt the rate with respect to the channel quality, and reaches 5,5 and 11 Mbit/s, depending on the code's

[6]The Signal to Noise Ratio (SNR) is defined as the ratio between the signal and the noise's power. It provides an intuitive measure for describing the fidelity with which the demodulation process in the receiver recovers the message signal from the modulated signal in the presence of additive noise [17].

[7]The Industrial, Scientific and Medical band is a license free portion of the spectrum around 900MHz and 2,4GHz. Because it's unlicensed, it's also crowded and subject to high interference.

[8]Quality of Services

length in use[9]. Also shorter physical layer frame and DQPSK modulation limit the overhead and increase the throughput. To guarantee compatibility through different releases, the preamble and header part of the physical frame are always transmitted at 1 Mbit/s.

**802.11a** Described as a clause of the 1999 specification, it allows to reach up to 54 Mbit/s using a different portion of the spectrum, the 5 GHz band, and a different transmission's technique: the Orthogonal Frequency Division Multiplexing (OFDM). This technique divide the main data stream in different data streams with lower rate and transmits every "sub-stream" at different carriers. These sub-carriers are characterized by orthogonality, that means that different carriers don't interfere each other. Moreover the modulation techniques are different, and based on the quality of the channel: for a good quality channel 64-QAM modulation can be used which guarantees up to 54 Mbit/s.

**802.11g** Ratified in June 2003 this extension is a "copy" of the 802.11a specification translated in the 2,4 GHz ISM band. As this is the same frequency of the 802.11b release, there are some compatibility issues to be solved due to different rates and techniques ( 802.11g uses OFDM that is a multicarrier technique while 802.11b uses DSSS, a single carrier technique). To guarantee compatibility while data are transmitted with OFDM, control and signaling information use the old physical layer based on modulation on a single carrier. Like 802.11a, even 802.11g transmits at a bit rate of 54 MBit/s.

**802.11n** This standard has the peculiarity of not being implemented for a specific portion of the spectrum, but can be used in different frequencies depending on the region. Just like other releases, the aim of this specification is to improve the throughput. To obtain this purpose 802.11n implements modification in physical and MAC layer, defining a new Advanced MAC layer [10]. In the Physical layer the increasing of the rate is obtained by the combination of different approaches like spatial multiplexing - the use of more antennas (MIMO - Multiple Input Multiple Output) in transmission and/or reception to have more independent information streams at once (up to 4 in 802.11n); A higher bandwidth - 40 MHz instead of 22 MHz; Enhanced modulation techniques with higher constellation, like 128,

[9] With respect to DSSS that uses barker sequences of 11 bit, CCK uses codes of 4 or 8 bit, to obtain respectively 5,5 and 11 Mbit/s

[10] More information about MAC layer in the next section, however Advanced MAC will not be covered for the purposes of this thesis.

256 QAM; Shorter guard bands. The combination of these improvements allow a theoretical rate of 600 Mbit/s, and commercial rate of 300 Mbit/s.

**802.11ac** Published in December 2013, is based on the previous standard and uses the 5 GHz spectrum. MIMO system now uses up to 8 antennas, and a MU-MIMO system [11] can be implemented. Also wider bandwidth up to 80 and 160 MHz are used. The theoretical bit-rate is 1 Gbit/s, while the real is 500 Mbit/s. This standard, as 802.11a, is less important in this work of thesis, because the implementation uses only 802.11b/g/n, in the 2,4 GHz bands.

Table 3.2 sums up all the standards and their principal characteristics.

| **Standard** | **Year** | **Band** | **Bandwidth** | **Modulation** | **Data Rate** |
|---|---|---|---|---|---|
| 802.11b | 1999 | 2,4 GHz | 20 MHz | CCK | 11Mb/s |
| 802.11a | 1999 | 5 GHz | 20 MHz | OFDM | 54Mb/s |
| 802.11g | 2003 | 2,4 GHz | 20 MHz | OFDM | 54Mb/s |
| 802.11n | 2009 | 2,4-5 GHz | 20-40 MHz | OFDM (up to 64-QAM) with up to 4 MIMO antennas | 600Mb/s |
| 802.11ac | 2013 | 5 GHz | 40-80-160 MHz | OFDM (up to 256-QAM) with up to 8 MIMO antennas | 1Gb/s |

*Table 3.2: 802.11 Amendments*

### 3.1.2 Medium Access Control layer - 802.11 MAC Frame

This section presents the MAC frame and IEEE 802.11 functions. Channel access control mechanism are not discussed here because are less important to the thesis' objective, but further information about 802.11 Medium Access Control can be found in [13].

The IEEE 802.11 syntax is more complex with respect to the wired LAN one, mainly because of the propagation medium that require more robust transmissions due to interference/attenuation.
There are three types of frame: Management, Control and Data frame.

[11] MU-MIMO is a technology that allow the router to communicate with multiple devices simultaneously, as opposed o MIMO with uses multiple antennas to communicate only to a single device at once.

Management frames are those type of frame used for functions like authentication, association, and for all those that enables stations to establish and maintain communications. Control frame coordinates the data transfers between stations and AP e.g. access control mechanism. The data frame, as the name says, transports the user data. The structure of the MAC frame is represented in the figure 3.4, along with the size of every field of the frame.

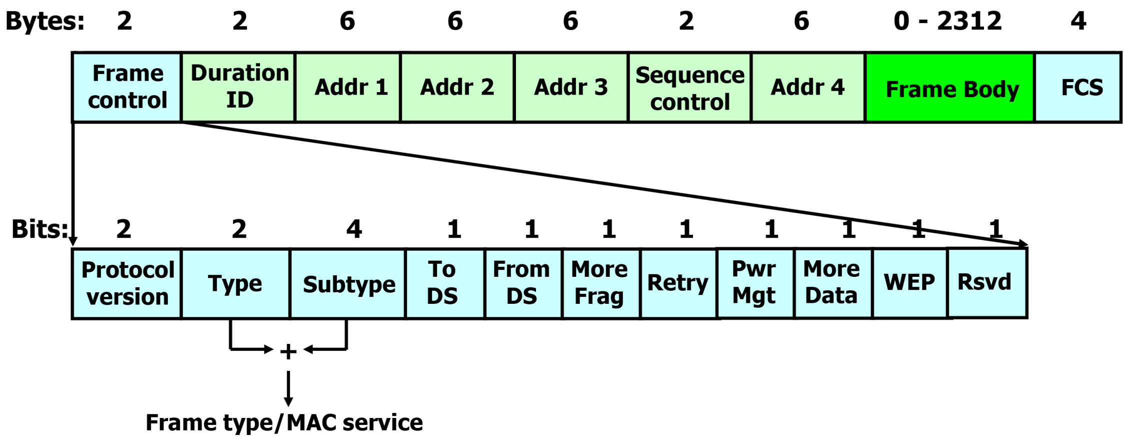


*Figure 3.4: IEEE 802.11 Physical and MAC Frame*

**Frame Ctrl** 2 Byte field that contain:

- *Protocol Version*: specifies the protocol version (the current is the 0)
- *Type*: 00 for Management, 01 for control and 10 for data frames. 11 is not assigned yet.
- *Subtype*: further information about the type of frame e.g. Probe Requests (fig. 3.6) are 0100.
- *To DS*: with the next field, specifies how to solve the 4 address fields.
- *From DS*: same as above field.
- *More Frag*: it's a flag with value 1 if there are more frame following.
- *Retry*: define if the frame is the retransmission of a previous frame.
- *Pwr Mgt*: assumes 1 value if the station goes to power safe mode after transmission.
- *More Date*: signal to the station in power safe that there are more data frame in the AP buffer.
- *Wep*: no more in use. It was used to indicate wep cryptography un the frame body.
- *RSVD*: indicates that frames have to be elaborated in receiving order.

**Duration ID** Indicate the transmission duration in microseconds.

**Address 1-4** Indicate the station, access point, source or destination based on ToDS and FromDS.

**Seq. Ctrl** used to represent the order of the fragments of the same frame and recognize duplicates.

**Frame Body** Has variable length and contain the informations of the upper level protocols.

**CRC** Cyclic Redundancy Check, it's an error code recognition sequence.

### 3.1.3 IEEE 802.11 Standard Scanning Functions

The IEEE 802.11 Standard implements various function such as addressing,power management, synchronization, authentication; for the purpose of this work however, we only focus on the *scanning function.*
Whenever a station wants to connect to an access point, it has to make an operation to discover nearby available BSSs. This operation is called *Scanning*, and can be perform in two different ways:

**Passive Scanning** The station selects the first channel and waits the beacon frames[12] from APs. Then changes channel and repeats the procedure until all the channels are scanned. This procedure, although less consuming because the station doesn't transmits anything, is slow and may require many seconds to be completed.

**Active Scanning** In this case the station always scans all the channels, but a solicitation mechanism is introduced: the station now participates actively to the scanning sending a packet called "Probe Request". As shown in figure 3.6 the

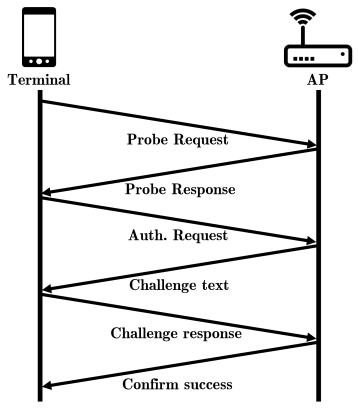


*Figure 3.5: Active Scanning and Authentication*

[12]Beacon frame is a management frame transmitted periodically to announce the presence of a Wireless LAN, and it contains all the information about it. In an Infrastructure BSS only the AP sends beacon frame.

station waits until the channel is free, then sends a probe request packet in broadcast[13]; Once a probe is sent the station starts a probe timer countdown, and waits for answers. At the end of the timer the station process the answers and then switches to the next channel. The APs that receive the probe request messages answer with a unicast[14] packet called "Probe Response", that contains information about its BSS. Probe responses are collected by the devices for every channel. This method is more power consuming, because requires the device to actually transmits packets and be active during the scanning period, but has the advantage of being faster than the passive method.

#### 3.1.3.1 Probe Requests - A little focus

The probe request is a packet of management type (0100 in the subtype field) sent by a station searching for available networks. Probe requests are a big hazard with respect to privacy, because they're not ciphered, hence the source and destination's MAC address are exposed to everyone who's monitoring the network. With the aid of a wireless card with sniffing capabilities any device in the connection range can captures probe requests from the surrounding environment, and keeps trace of different MAC addresses (for various purposes, see chapter 2).
Figure 3.6 shows the probe request frame.

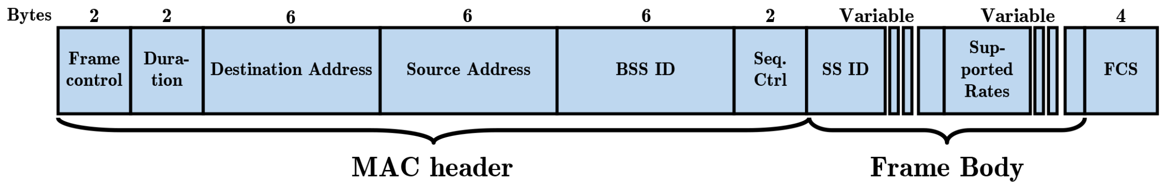


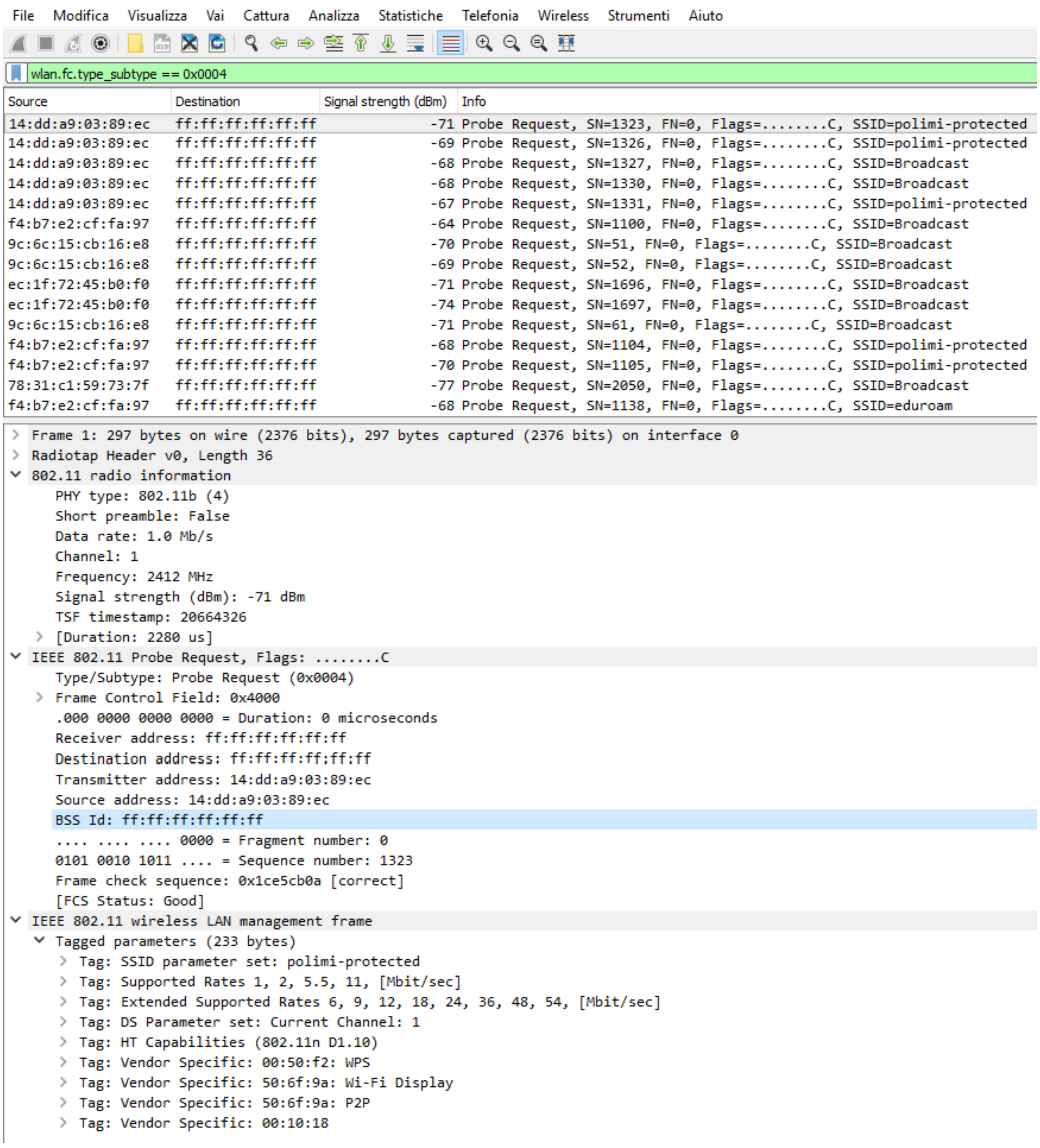
*Figure 3.6: Probe Request frame's structure*

Some important fields for our purposes are:

- Destination Address: is the broadcast MAC address FF:FF:FF:FF:FF:FF
- Source Address: is the device's MAC address
- SSID: it's the network's name e.g. "polimi"
- Supported Rates: List of the rates supported by the device
- RSSI: Received signal strength indication is the power level in dBm at which the sniffing adapter received the packet. Gives also an estimate of the distance of the device.

[13]Broadcast's destination address is FF:FF:FF:FF:FF:FF, and it means that all the devices in that range are recipient of the message.
[14]Unicast means that the message have a unique recipient

In figure 3.7, thanks to the aid of a software called Wireshark[15], a probe packet structure can be seen.



```
wlan.fc.type_subtype == 0x0004
```

```
Source             Destination        Signal strength (dBm)  Info
14:dd:a9:03:89:ec  ff:ff:ff:ff:ff:ff  -71 Probe Request, SN=1323, FN=0, Flags=........C, SSID=polimi-protected
14:dd:a9:03:89:ec  ff:ff:ff:ff:ff:ff  -69 Probe Request, SN=1326, FN=0, Flags=........C, SSID=polimi-protected
14:dd:a9:03:89:ec  ff:ff:ff:ff:ff:ff  -68 Probe Request, SN=1327, FN=0, Flags=........C, SSID=Broadcast
14:dd:a9:03:89:ec  ff:ff:ff:ff:ff:ff  -68 Probe Request, SN=1330, FN=0, Flags=........C, SSID=Broadcast
14:dd:a9:03:89:ec  ff:ff:ff:ff:ff:ff  -67 Probe Request, SN=1331, FN=0, Flags=........C, SSID=polimi-protected
f4:b7:e2:cf:fa:97  ff:ff:ff:ff:ff:ff  -64 Probe Request, SN=1100, FN=0, Flags=........C, SSID=Broadcast
9c:6c:15:cb:16:e8  ff:ff:ff:ff:ff:ff  -70 Probe Request, SN=51, FN=0, Flags=........C, SSID=Broadcast
9c:6c:15:cb:16:e8  ff:ff:ff:ff:ff:ff  -69 Probe Request, SN=52, FN=0, Flags=........C, SSID=Broadcast
ec:1f:72:45:b0:f0  ff:ff:ff:ff:ff:ff  -71 Probe Request, SN=1696, FN=0, Flags=........C, SSID=Broadcast
ec:1f:72:45:b0:f0  ff:ff:ff:ff:ff:ff  -74 Probe Request, SN=1697, FN=0, Flags=........C, SSID=Broadcast
9c:6c:15:cb:16:e8  ff:ff:ff:ff:ff:ff  -71 Probe Request, SN=61, FN=0, Flags=........C, SSID=Broadcast
f4:b7:e2:cf:fa:97  ff:ff:ff:ff:ff:ff  -68 Probe Request, SN=1104, FN=0, Flags=........C, SSID=polimi-protected
f4:b7:e2:cf:fa:97  ff:ff:ff:ff:ff:ff  -70 Probe Request, SN=1105, FN=0, Flags=........C, SSID=polimi-protected
78:31:c1:59:73:7f  ff:ff:ff:ff:ff:ff  -77 Probe Request, SN=2050, FN=0, Flags=........C, SSID=Broadcast
f4:b7:e2:cf:fa:97  ff:ff:ff:ff:ff:ff  -68 Probe Request, SN=1138, FN=0, Flags=........C, SSID=eduroam
```

```
> Frame 1: 297 bytes on wire (2376 bits), 297 bytes captured (2376 bits) on interface 0
> Radiotap Header v0, Length 36
v 802.11 radio information
    PHY type: 802.11b (4)
    Short preamble: False
    Data rate: 1.0 Mb/s
    Channel: 1
    Frequency: 2412 MHz
    Signal strength (dBm): -71 dBm
    TSF timestamp: 20664326
  > [Duration: 2280 us]
v IEEE 802.11 Probe Request, Flags: ........C
    Type/Subtype: Probe Request (0x0004)
  > Frame Control Field: 0x4000
    .000 0000 0000 0000 = Duration: 0 microseconds
    Receiver address: ff:ff:ff:ff:ff:ff
    Destination address: ff:ff:ff:ff:ff:ff
    Transmitter address: 14:dd:a9:03:89:ec
    Source address: 14:dd:a9:03:89:ec
    BSS Id: ff:ff:ff:ff:ff:ff
    .... .... .... 0000 = Fragment number: 0
    0101 0010 1011 .... = Sequence number: 1323
    Frame check sequence: 0x1ce5cb0a [correct]
    [FCS Status: Good]
v IEEE 802.11 wireless LAN management frame
  v Tagged parameters (233 bytes)
    > Tag: SSID parameter set: polimi-protected
    > Tag: Supported Rates 1, 2, 5.5, 11, [Mbit/sec]
    > Tag: Extended Supported Rates 6, 9, 12, 18, 24, 36, 48, 54, [Mbit/sec]
    > Tag: DS Parameter set: Current Channel: 1
    > Tag: HT Capabilities (802.11n D1.10)
    > Tag: Vendor Specific: 00:50:f2: WPS
    > Tag: Vendor Specific: 50:6f:9a: Wi-Fi Display
    > Tag: Vendor Specific: 50:6f:9a: P2P
    > Tag: Vendor Specific: 00:10:18
```

*Figure 3.7: Probe Request Frame - Wireshark*

As seen in Chapter 2, since the probe requests are unencrypted and contain the unique ID of the device, they can be collected and analysed in order to tracking devices for various scopes e.g. targeted advertising. In 2013, for example, an article from Ars Technica [18] reports a known marketing firm which implements Wi-Fi probe tracking

[15]https://www.wireshark.org

behind a smart trash can.
In order to avoid (or better raise the complexity) user-tracking through Wi-Fi probe detection, most nowadays Operating Systems (OSs) have implemented different variants of MAC address randomization. This technique rely upon the transmission of probe requests with different MAC addresses with respect to the real device's one. The effect is that when the device is in scanning mode, it uses a random ID for every packet, so the user's tracking is more complex, and therefore privacy less compromised.
Since the purpose of this work is the occupancy estimation through Wi-Fi probe detection, the MAC address randomization is a real problem, because it leads to an overestimation of the number of devices. To our advantage, only iOS devices actually used this technique by default.
The method used to solve this problem is shown in the next chapter, but it is important to point out that our device doesn't send out the collected MAC addresses, but the sole total number of them.

## 3.2 The MQTT Protocol

In 1999 Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom) needed a communication protocol that minimizes the power and band consumption during device communication. This protocol should have been simple to implement, reliable, low data consuming and agnostic[16]. What was conceived for internal use was then published without license costs in 2010 with the name Message Queue Telemetry Transport. In 2014 the MQTT was standardized by the OASIS[17], and in 2016 by the ISO[18] with name ISO/IEC PRF 20922. Nowadays MQTT is used for Machine-to-Machine communication, Internet of Things and every environment that requires simple implementation and low power consumption. Big corporation like Facebook (with Facebook Messenger) and Microsoft (with Azure IoT) use the MQTT protocol.

### 3.2.1 Protocol Structure

MQTT is based on a publish/subscribe model: a client (could be a sensor, computer, smartphone etc.) called "publisher" sends a message, and other clients called "subscribers" can receive this message (one-to-many message distribution system). In this way publisher and subscriber ignore the existence of other clients. Moreover a third player named "Broker" is added. It behaves as a sort of "client proxy", and is able to filter and distribute the communication between publishers and subscribers - it manages the data stream. With the term "client" we intend both publisher and subscriber. This infrastructure doesn't need any synchronization, and an example can be see in figure 3.8.

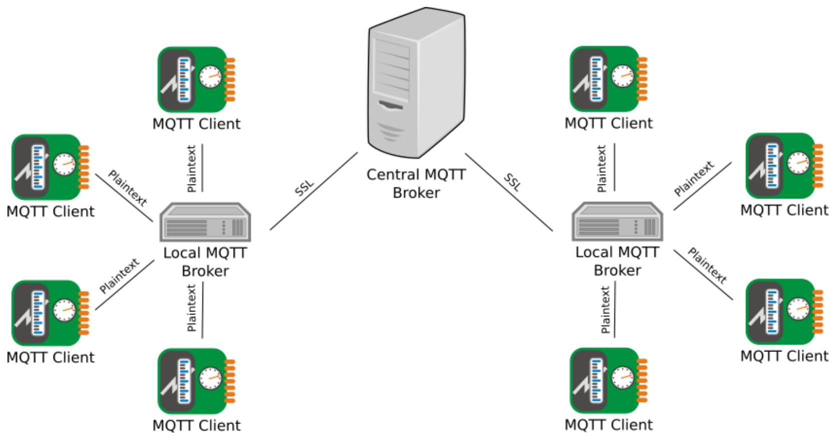


*Figure 3.8: An example of MQTT structure*

[16]Agnostic, in computer terms, means independent from the implementation's platform.
[17]Organization for the Advancement of Structured Information Standards, see [10]
[18]International Organization for Standardization, see [6]

**Broker** is the main player of the publish/subscribe structure. Its role is to manage all the mqtt clients and their messages, then filters and decides which subscribers has to receive them or not.

**MQTT client** can be a publisher, subscriber or both and sends or receives messages from the MQTT broker.

**Topic** is an UTF-8 string[19] with a structure similar to directories. This string is used by the broker to filter and sort incoming messages, and by the client to publish and subscribe. The portions of the topic are separated by a slash "/" called "topic level separator". An example could be:

$$Polimi/DEIB/floor3/office5/temperature \tag{3.1}$$

Topics are case-sensitive and also spaces can be used, so "Polimi" its different from "polimi" and "hithere" its different from "hi there" topic. In order to enhance filtering capabilities mqtt implements some "wildcards", for example:

$$Polimi/DEIB/floor3/+/temperature \tag{3.2}$$

$$Polimi/DEIB/floor3/\# \tag{3.3}$$

Topic 3.2 shows the use of a single level wildcard (+) to show all the temperature data for the third floor of the DEIB building, while 3.3 topic uses a multi level wildcard (#) that shows all the data (e.g. temperature, humidity, occupancy) of the same place. Wildcards are useful to select more topics at the same time.

**Filters** The broker filters messages so that any subscriber receives only the right ones. This filtering procedure can be based on different properties:

- Topic: as seen above, the broker can filter and sort messages with respect to the topic they refer, and delivers only the required topic's messages.
- Content: the message is filtered based on its content. It's a powerful but also risky method because messages can change or be ciphered.
- Type: interesting method that can, for example, recognize and filter error messages from event messages.

[19]UTF-8 is a character encoding standard, like ASCII, to represent symbols.

### 3.2.2 Retain function and QoS

When a client subscribe to a topic, this last one is empty. Only when someone publish something the subscribers will see the change. Thanks to the retain function a publisher can force the broker to save the last message for a given topic. In this way when a new client subscribes to that topic, the broker sends the last saved message. This could be useful, for example, in the case in which a client subscribes a temperature topic which receives updates every 15 minutes. Thanks to the retained flag by the publisher side, the subscriber receives the last temperature registered in the room without having to wait for the next update.
The basic publish/subscribe model described doesn't guarantees that a sent message arrives at destination. To improve the reliability of the system QoS techniques where introduced, in particular the standard distinguishes among 3 different class of Quality of Service:

**Class 0** "At most one": is a best effort approach without any warranty, that means that no acknowledge of successful transmission packet is needed by transmission's side.

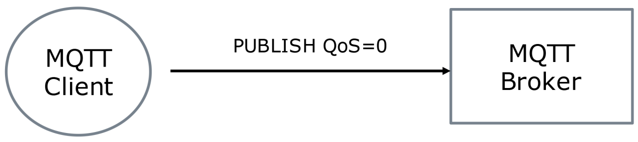


*Figure 3.9: QoS=0 transmission*

**Class 1** "At least one": the client continues to transmit the message until it is acknowledged (PUBACK) by the broker. In this case the message can be received more that once.

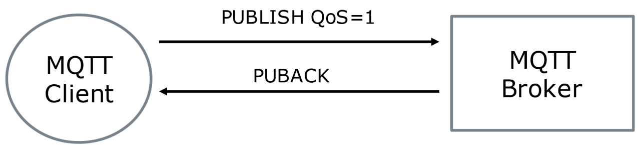


*Figure 3.10: QoS=1 transmission*

**Class 2** "Exactly once": the client sends the message with QoS field equal to 2. The broker receives the message and stores its ID while sends a PUBREC packet to the client to inform it about the correct reception. The storing of the message ID is useful to the broker to discard duplicate packets. When the client receives

the PUBREC discards the initial packet and sends a PUBREL. Upon receiving the PUBREL packet the broker clears the stored ID and ends the exchange with a PUBCOMP message.
With this method the message arrives only once.

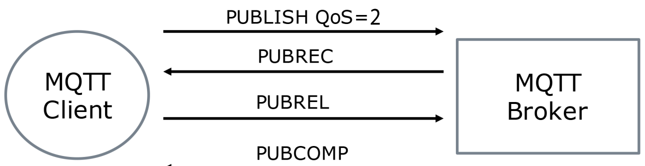


*Figure 3.11: QoS=2 transmission*

### 3.2.3 Protocol Messages and Example

This section is intended as a review of the type of packets of MQTT protocol, and an example of connection, publish and subscribe.
Table 3.3 shows some control packets and their description.

| **Control packet** | **Direction of Flow** | **Description** |
|---|---|---|
| CONNECT | Client to Server | Client request to connect to server |
| CONNACK | Server to Client | Connect acknowledgement |
| PUBLISH | Client to Server or Server to Client | Publish message |
| PUBACK | Client to Server or Server to Client | Publish acknowledgement |
| PUBREC | Client to Server or Server to Client | Publish received (QoS=2 part 1) |
| PUBREL | Client to Server or Server to Client | Publish release (QoS=2 part 2) |
| PUBCOMP | Client to Server or Server to Client | Publish complete (QoS=2 part 3) |
| SUBSCRIBE | Client to Server | Client subscribe request |
| SUBACK | Server to Client | Subscribe acknowledgement |
| UNSUBSCRIBE | Client to Server | Unsubscribe request |
| UNSUBACK | Server to Client | Unsubscribe acknowledgement |
| PINGREQ | Client to Server | PING request |
| PINGRESP | Server to Client | PING response |
| DISCONNECT | Client to Server | Client is disconnecting |

*Table 3.3: MQTT protocol - messages*

**Connect**

The connection starts with a request called *Connect*, composed by essential informations like the client ID, the type of connection - *session* if the broker has to save all client's activity and loss packets; *Clean* if all the informations of previous session are cleaned. Also a username and password fields are required to authenticate the client; it's possible to use SSL encryption and in this case username and password are not necessary. It is also possible to send a "last will message", that informs the other clients in case of an interruption.
The broker responds with a *CONNACK* packet that contains the session, a flag indicating if the opening of the session is successful (if the session was already open the flag will be false) and a return code from 0 to 5 that tells the result of the action:

- **0**: Successful connection
- **1**: Unacceptable version
- **2**: ID rejected
- **3**: Server unavailable
- **4**: Wrong username and password
- **5**: Unauthorized

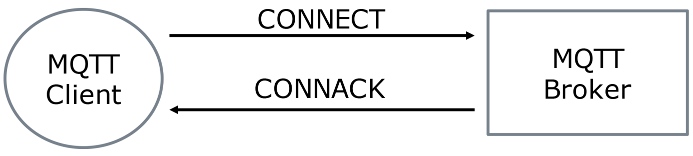


*Figure 3.12: Connection scheme*

**Publish**

The publish message is composed by a packet identifier (a name or unique code), the topic, the type of QoS required, the retain flag, the payload (the data of the packet) and a final flag that tells if the packet is a retransmission or not. The respond to the *PUBLISH* message depends on the QoS field as seen above.

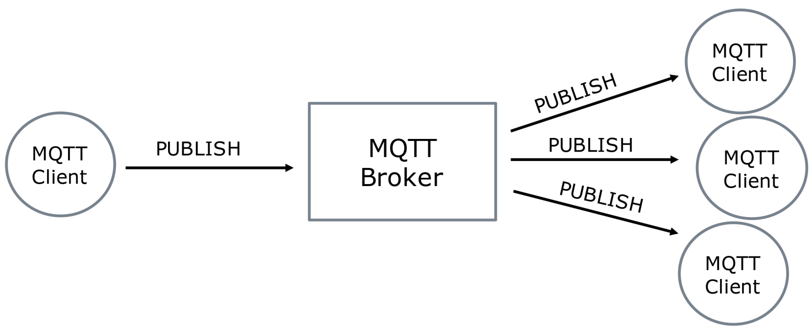


*Figure 3.13: Publish scheme*

### Subscribe

This is the message that the receiving client sends to the broker in order to announce his availability to receive data from the respected publisher/topic. In a single subscribe message the subscriber must specify the topic/s (a single subscription message can contain more than one topic) and QoSs requirements. This message is acknowledged by a *SUBACK* message that must contain the packet ID used by the *SUBSCRIBE* packet, and a return code like *CONNECT* messages' ones, for each topic subscribed.

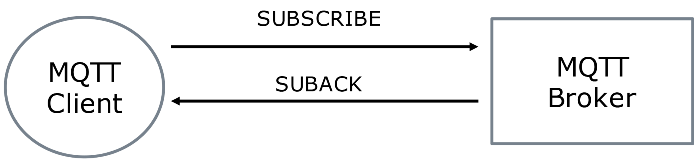


*Figure 3.14: Subscribe scheme*

### Unsubscribe

This message cancels the client's subscription to a topic, so it contains the client ID and the topic from which being disconnected. As a response the broker sends an *UNSUBACK* message containing only the packet ID used by the *UNSUBSCRIBE* message.

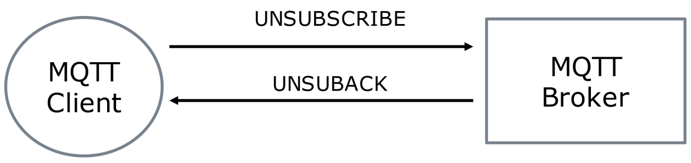
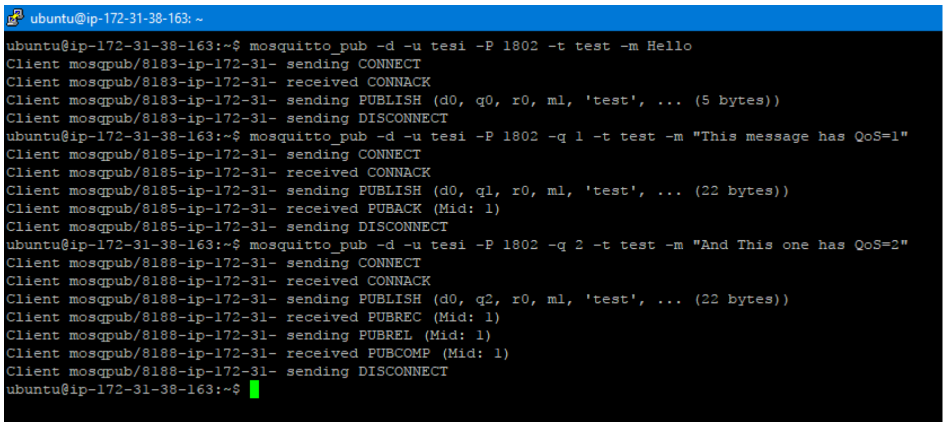


*Figure 3.15: Unsubscribe scheme*

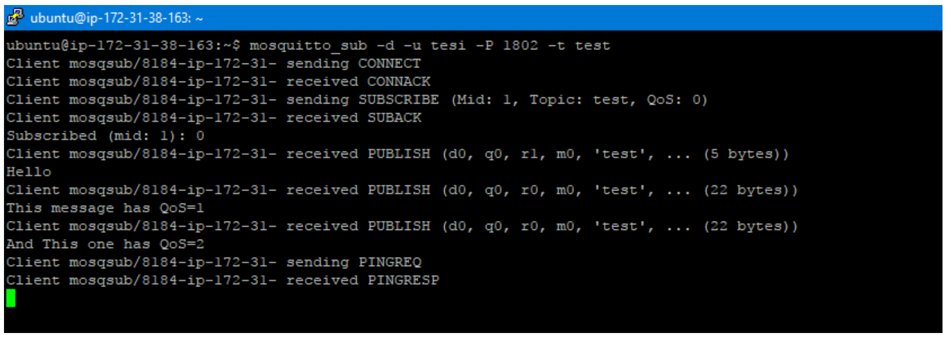
Figure 3.16 shows an example of publication[20] to a topic "test" with different QoS for each message.



```
ubuntu@ip-172-31-38-163: ~
ubuntu@ip-172-31-38-163:~$ mosquitto_pub -d -u tesi -P 1802 -t test -m Hello
Client mosqpub/8183-ip-172-31- sending CONNECT
Client mosqpub/8183-ip-172-31- received CONNACK
Client mosqpub/8183-ip-172-31- sending PUBLISH (d0, q0, r0, m1, 'test', ... (5 bytes))
Client mosqpub/8183-ip-172-31- sending DISCONNECT
ubuntu@ip-172-31-38-163:~$ mosquitto_pub -d -u tesi -P 1802 -q 1 -t test -m "This message has QoS=1"
Client mosqpub/8185-ip-172-31- sending CONNECT
Client mosqpub/8185-ip-172-31- received CONNACK
Client mosqpub/8185-ip-172-31- sending PUBLISH (d0, q1, r0, m1, 'test', ... (22 bytes))
Client mosqpub/8185-ip-172-31- received PUBACK (Mid: 1)
Client mosqpub/8185-ip-172-31- sending DISCONNECT
ubuntu@ip-172-31-38-163:~$ mosquitto_pub -d -u tesi -P 1802 -q 2 -t test -m "And This one has QoS=2"
Client mosqpub/8188-ip-172-31- sending CONNECT
Client mosqpub/8188-ip-172-31- received CONNACK
Client mosqpub/8188-ip-172-31- sending PUBLISH (d0, q2, r0, m1, 'test', ... (22 bytes))
Client mosqpub/8188-ip-172-31- received PUBREC (Mid: 1)
Client mosqpub/8188-ip-172-31- sending PUBREL (Mid: 1)
Client mosqpub/8188-ip-172-31- received PUBCOMP (Mid: 1)
Client mosqpub/8188-ip-172-31- sending DISCONNECT
ubuntu@ip-172-31-38-163:~$
```

*Figure 3.16: Publication with PuTTY and a Mosquitto broker*

Figure 3.17 shows a subscription to the same topic where the subscriber sees what the publisher sends.



```
ubuntu@ip-172-31-38-163: ~
ubuntu@ip-172-31-38-163:~$ mosquitto_sub -d -u tesi -P 1802 -t test
Client mosqsub/8184-ip-172-31- sending CONNECT
Client mosqsub/8184-ip-172-31- received CONNACK
Client mosqsub/8184-ip-172-31- sending SUBSCRIBE (Mid: 1, Topic: test, QoS: 0)
Client mosqsub/8184-ip-172-31- received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/8184-ip-172-31- received PUBLISH (d0, q0, r1, m0, 'test', ... (5 bytes))
Hello
Client mosqsub/8184-ip-172-31- received PUBLISH (d0, q0, r0, m0, 'test', ... (22 bytes))
This message has QoS=1
Client mosqsub/8184-ip-172-31- received PUBLISH (d0, q0, r0, m0, 'test', ... (22 bytes))
And This one has QoS=2
Client mosqsub/8184-ip-172-31- sending PINGREQ
Client mosqsub/8184-ip-172-31- received PINGRESP
```

*Figure 3.17: Subscription with PuTTY and a Mosquitto broker*

In this cases either publication and subscription are protected by username and password: in this way not all the users that knows the broker's public IP address and topic can publish or subscribe because they have to know the credentials (in this case "tesi" and "1802").

[20]See www.mosquitto.org for more details about the commands

## 3.3 Node-RED and MQTT duo

In the 1970s J. Paul Morrison, computer programmer for IBM, invented the flow-based programming as a way of describing an application's behaviour with a series of black-boxes connected together to form a network. Each box has a well-defined purpose, with a data input, an elaboration part and an output. In this way the representation of a program in simpler because each problem can be divided in simple steps and, even without understand every line of code within each box, everyone can look at the glow and get a sense of what it's doing.
Based on the idea of flow-based programming, in early 2013 Nick O'Leary and Dave Conway-Jones, IBM's engineers, give birth to Node-RED[21].

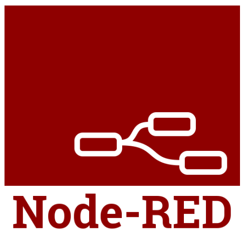


*Figure 3.18: The Node-RED logo*

Started as a tool for visualising and manipulating mappings between MQTT topics, Node-RED quickly became a much more general software tool for wiring together hardware devices, APIs and online services. In September 2013 Node-RED became Opensource and in October 2016 it became one of the founding projects of the JS Foundation.
Nowadays Node-RED provides a browser-based editor that makes it easy to wire together flow using the wide range of nodes in the palette. There are node for every type of service, and new nodes are developing everyday. The programming language used by Node-RED is JavaScript but the flow-implementation allows to reduce users' programming knowledge to a minimum.
Here are some examples of flows implementation with Node-RED:

1. The following (3.19) is a simple flow created using the *Dashboard* module that create a live data dashboard. The input nodes receive a numerical value (slider) and a string (text input) and send them via MQTT thanks to the MQT output node properly configured. Then two MQTT input nodes are configured to receive inputs from the same topic used as output before, and print their outputs to a

[21] See www.nodered.org for more details

free line space in the dashboard, and a graph. Also a ping node is used, and configured to ping Google's site and giving the result in the dashboard.

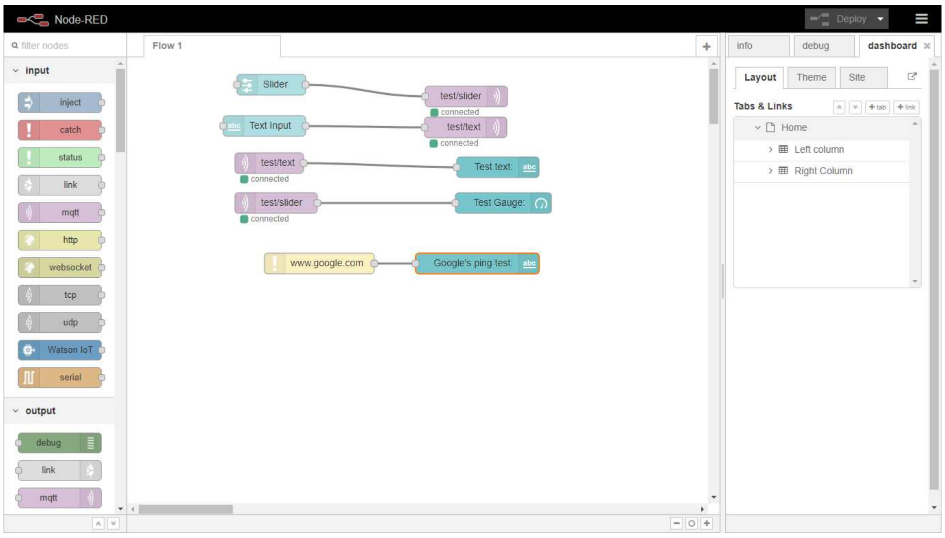


Figure 3.19: Node-RED flow page

2. The next figure (3.20) is the dashboard's user interface of the above flow (3.19). Slider, text input, text output and graph are shown in the dashboard UI, while MQTT input and output nodes are hidden because they're not from the dashboard module. The website created by Node-RED is visible, within the network, searching for the machine's IP address (the computer where Node-RED is installed) followed by :1880 (the default Node-RED's port).

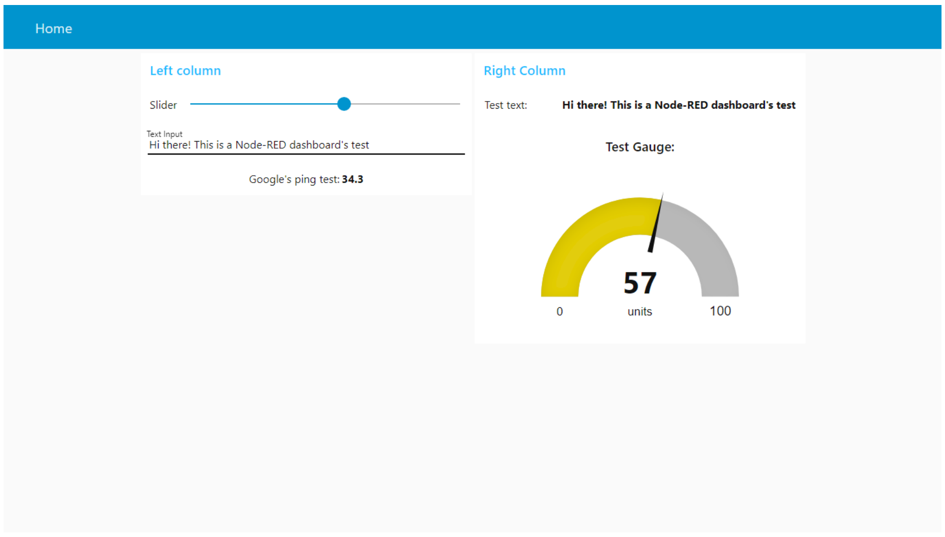


Figure 3.20: Node-RED dashboard module

3. The last figure show a more complex flow, divided in two main parts: the top part begins with an HTTP request node, that retrieves the current temperature that a sensor is reading via HTTP request. After that the flow proceed with a function node that filters out outlier values, those outside of a certain range. Finally if the temperature is determined to not be an outlier, it is stored into a database in the final node.
   The lower half part of the flow is settled to be triggered every 15 minutes. During this section the program looks at the data stored in the database and collects the average temperature over a 15 minute time frame. Once done, the average is sent via MQTT message and either turn on air conditioning or do nothing based upon a certain temperature threshold.

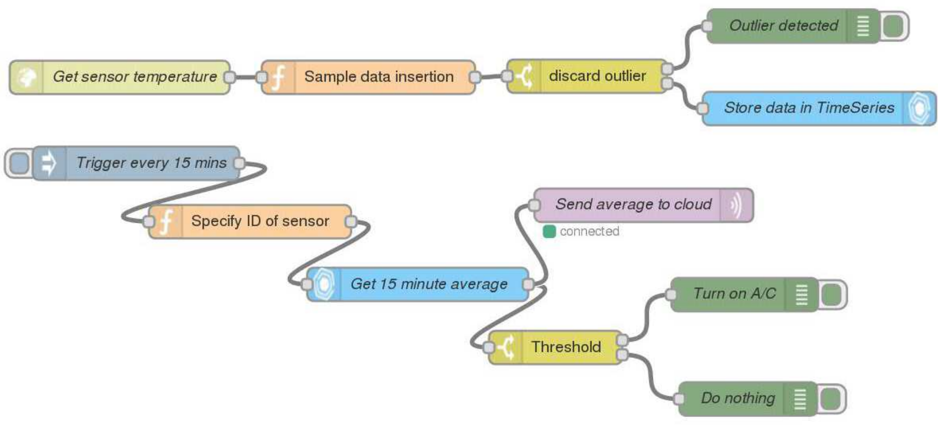


Figure 3.21: Node-RED flow

## 3.4 The ESP8266 Microcontroller

### 3.4.1 The Microcontroller

A microcontroller (also called MCU - MicroController unit) is an integrated electronic device on a single chip. It is born as an evolution of the microprocessor and it's used in embedded systems. The microcontroller, as opposed to the microprocessor, is a complete system that integrates on a single chip the processor, the program and RAM memory and also some I/O channels (Pins).
Since 2005 the microcontroller panorama has changed and gain more interest from all sort of people. Thanks to Arduino platform, the microcontrollers and in general the world of DIY (Do It Yourself) spread among all the world. This is because the simplicity of fast prototiping, where thanks to a hardware platform connected to the pc and a simple piece of software with which people can program microcontrollers, makes simple electronic projects and basic knowledge available to all the people.

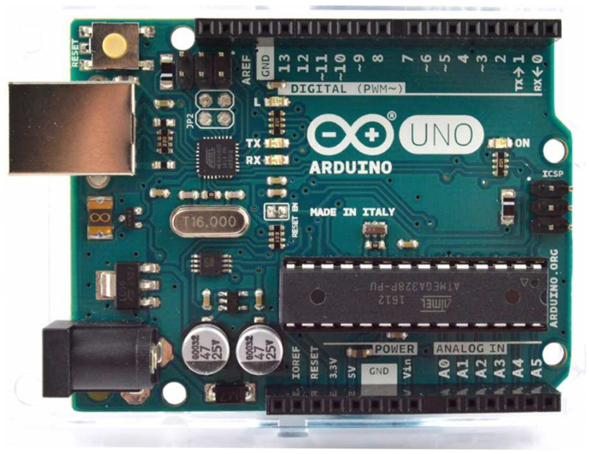


*Figure 3.22: Arduino Uno board*

Figure 3.22 shows an Arduino Uno board. It uses an ATmega328P microcontroller, 32KB of flash memory, 2KB of ram, 1KB EEPROM, 14 I/O digital pins and 6 analog input pins. Thanks to its simple hardware, wiring, and software, it's easy for hobbyist and professional prototyping with the board.
Thanks to its open-source nature, the Arduino has a lot of cheapest clones, that work with the same software and helped expanding the microcontroller diffusion. Along with clones, shields that expand Arduino Uno capabilities, like adding more I/O pins, Wi-Fi, Servo-motor dedicated connectors and so on, were commercialized.
One of this low cost "expansion" adds Wi-fi, and it's based on the ESP8266 chip by Espressif.

### 3.4.2 Espressif Systems' ESP8266

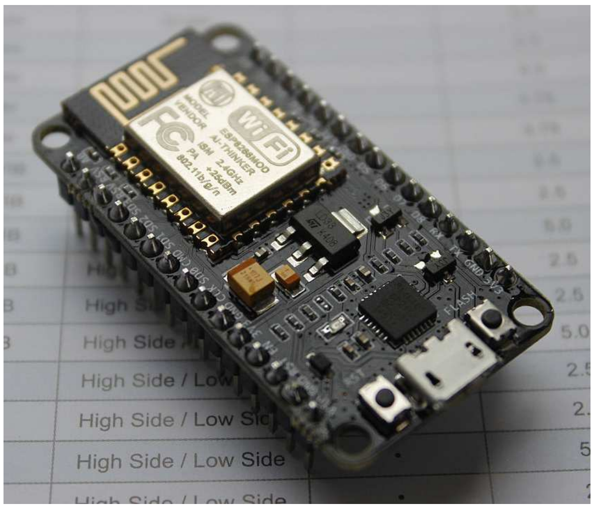


*Figure 3.23: NODEMCU devkit, based on ESP-12 module*

The ESP8266 is a low cost Wi-Fi chip produced by the Chinese manufacturer Espressif Systems. It combines Wi-Fi capabilities with an MCU with its memory, ram and I/O pins. Depending on the module's version it can have from 3 to 22 active pins. Its very low price (more or less 2$) and compatibility with Arduino IDE contribute to its diffusion. Some of the chip's features are:

**Power** ESP work at 3.3v, while Arduino works at 5v.

**CPU** Tensilica Xtensa Diamond Standard L106 Micro, a 32bit RISC microprocessor with a clock frequency of 80MHz.

**Memory** The NodeMCU module used in this thesis has 128 KB and 4 MB of respectively memory and storage.

**Connectivity** IEEE 802.11 b/g/n Wi-Fi, with support to WEP and WPA/WPA2 authentication.

**GPIO** 16 General Purpose I/O pins

**Interfaces** ESP support SPI, I$^2$C and UART on dedicated pins.

In this thesis the NODEMCU module with ESP-12 version is used. With respect to ESP-01 module, this implementation has the advantage of having an FTDI USB to serial UART integrated, and can be connected to the PC with a simple USB - microUSB cable, without having to connect other components.
Figure 3.24 shows the pin disposition and functions in a NODEMCU v1 board:

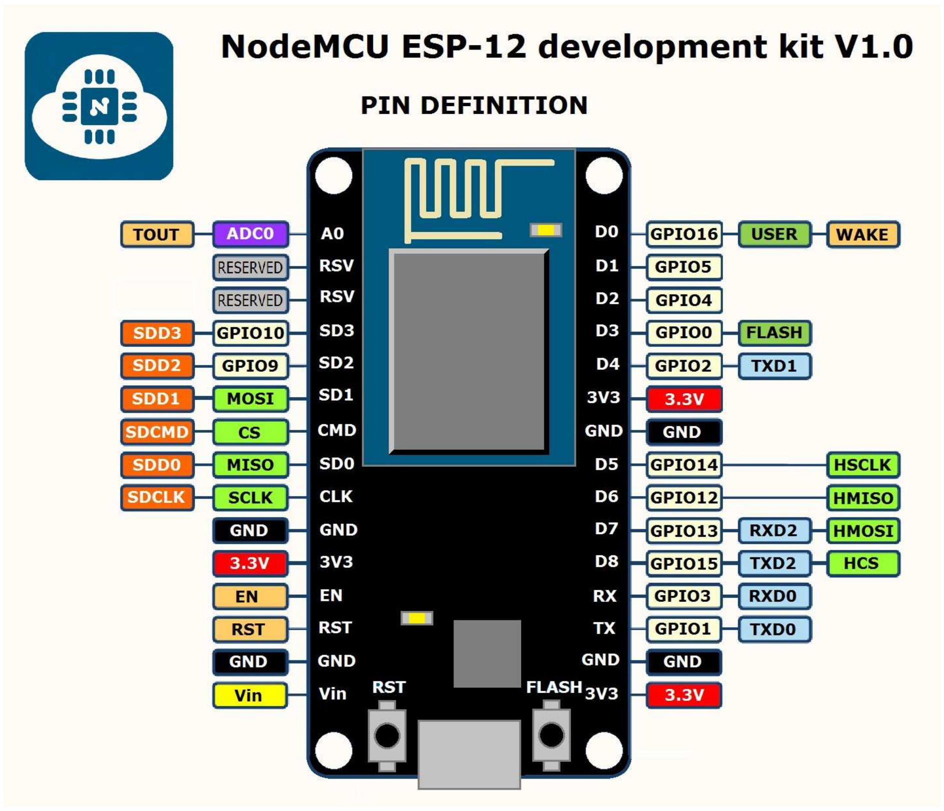


*Figure 3.24: NodeMCU pin arrangement*

#### 3.4.2.1 The Watchdog Timer

A common features of microcontrollers is the so-called Watchdog. The main reason to explain this part of the MCU will be seen later on. The watchdog timer is a timer used to detect infinite loops or deadlock during an execution. In figure 3.25 a graphical explanation can be seen.

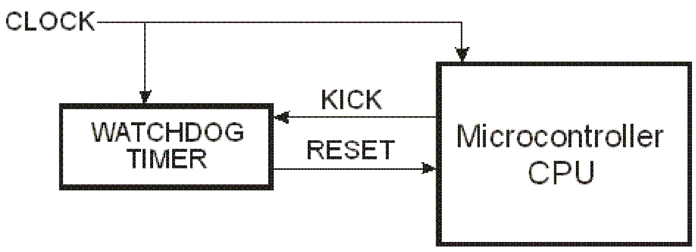


*Figure 3.25: Watchdog timer function*

During the normal execution of a program, the microcontroller regularly resets the watchdog timer, in order to avoid its expiration. If a malfunction, some long cycles or deadlock occurs, the MCU is not able to "kick" (or fed) the timer. When this one expires, the harware is automatically reset, in order to prevent the deadlock. The problem with this timer is that, in case of long cycles, it may expires and reset the chip even when not necessary.

## 3.4.3 Serial Interfaces

A serial interface is a communication interface that transmit data in serial mode, that means one bit at a time sequentially. There are a number of different standards for serial interfaces to microcontrollers, that differ from the number of pins and approaches to communication.
In this section we'll explore the $I^2C$ and SPI serial interfaces, in order to better comprehend the principal connections used with microcontrollers and the one used in this project.

### 3.4.3.1 $I^2C$

Sometimes known as the Two-Wire Interface (TWI) it is a bus (Binary Unit System) that can support multiple devices connected to the same two wires. It can run at either 5 or 3.3V, with top speeds of up to 400Kbit/s.
The two data lines of $I^2C$ operate as both inputs and outputs, and must have pull-up resistors connected (unless they are integrated in the microcontroller board). In general remote sensors connected to the microcontroller will require 4 wires: two for data and two for power ($V_{in}$ and GND). $I^2C$ is based on the master-slave paradigm: devices are either masters or slaves, and there can be more than one master device per bus.

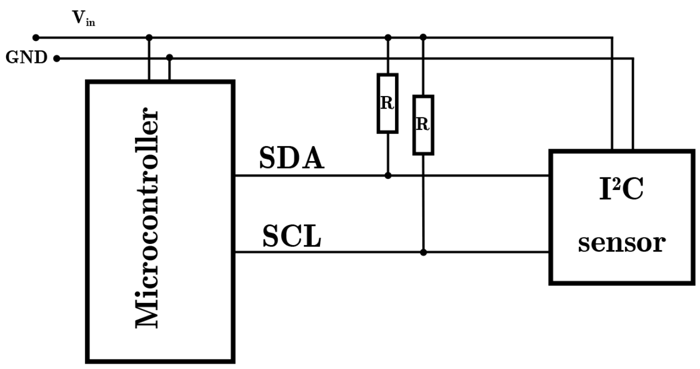


*Figure 3.26: An $I^2C$ connection*

The SCL is the Serial Clock Line, and it's a clock, the timing supplied by the master. The SDA is the Serial DAata line and carriers the data. When there is data to be transmitted, the sender (master or slave) takes the SDA line and sends data as logic hights or lows in time with the clock signal. When the transmission is complete the SDA pin is released.

#### 3.4.3.2 SPI

The Serial Peripheral Interface is a microcontroller bus standard that uses four data lines and a top speed of 80Mbit/s. While the first three data lines are shared, the fourth is unique for every device connected. In this interface there can only be one master device (usually a microcontroller).
Figure 3.27 shows an example of SPI connection.

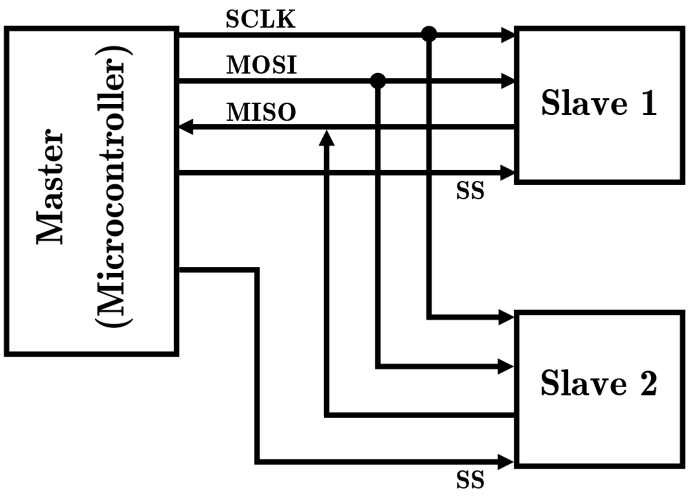


*Figure 3.27: This figure shows an example of SPI connection with 2 peripherals*

The master have a dedicated Slave Select (SS) line for each of the slave devices to select the one it communicates with. The other two lines are required for data communication since separate lines are used for each direction of communication:
MOSI - Master Out Slave In, carriers the data from the master to the slave device.
MISO - Master In Slave Out does the reverse, carriers the data from the slave to the master.
Finally the SCLK (Serial CLocK) line is for the clock.
The SPI protocol is also used as a means of ICSP (In-Circuit Serial Programming) on some microcontrollers such as the ATmega[22] and ATtiny[23] families.

[22] http://www.atmel.com/products/microcontrollers/avr/megaavr.aspx
[23] http://www.atmel.com/products/microcontrollers/avr/tinyavr.aspx

# Chapter 4

# System Implementation

In every project there are theoretical basis and a practical application of them. In the previous chapter the theoretical fundamentals were discussed while this chapter is intended as an explanation of the system's implementation. In order to avoid messing stuff up, and divide what is the hardware implementation from the software one, the chapter is divided into two main parts:

1. Sensor-side: the "hardware part" of the project (except for the program inside the ESP8266) dedicated mainly to the construction of the circuit. In this part the choices, characteristics and functionalities of the sensors are discussed. After a brief introduction on their characteristics and behaviours, the schematic drawing is illustrated with the building of the prototype. Later on the PCB[1] layout and making process is faced, in order to implement a more reliable and compact hardware device. In conclusion the software part of the microcontroller is discussed: its functions, behaviour, implementation issues that came up and the final code with a flowchart explanation.

2. Server-side: it's the "software part" of the project. This part will cover the server set-up, started with a Raspberry Pi 3, then with Amazon Web Services and at the end with Polimi Cloud. The Mosquitto MQTT broker installation, with some test commands and configuration file examples. Node-RED installation, its modules and flow implementation and, in the conclusion, a little hint of the telegram's application role, and its bot implementation in Node-RED.

Acquired data and their analysis are discussed in the sequent chapter, with the aid of MATLAB programming and simulations tools.

[1]Printed Circuit Board is a mechanical support for electronic components that connects them using conductive tracks etched from copper-covered boards.

## 4.1 Sensor-Side Implementation

### 4.1.1 Sensors and wiring

The main purpose of the project was to implement a system that gives occupancy information using Wi-Fi probe requests, along with ambient informations like temperature, humidity and pressure.
Starting from this assumptions we chose the NodeMCU ESP8266 chip that (see chapter 3 section 3.4.2) is based on the esp-12 chip, and have Wi-Fi capabilities and an integrated micro-controller with some I/O pins. On the I$^2$C pins of the ESP (pin D1 and D2) we connect Adafruit's environment sensor based on the Bosch BME280 (fig.4.1).

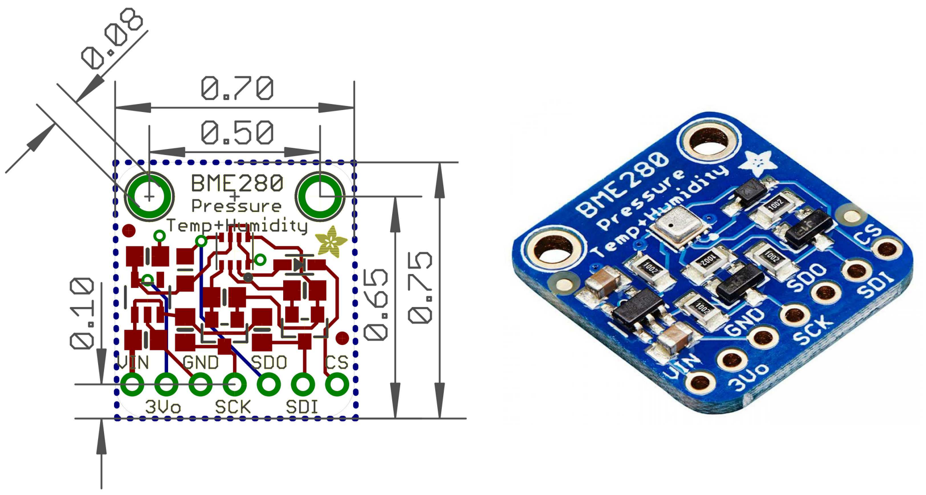


*Figure 4.1: Adafruit BME280 PCB and sensor's picture*

This sensor is capable of detect temperature, humidity and barometric pressure with an accuracy of ±1°C, ±3% and ±1hPa respectively. Given that pressure change with altitude, the sensor can also be used as an altimeter, with an accuracy of ±1 meter or less. As a matter of fact, given the air pressure at sea level, between altitude (h) and pressure (p) exists the relation:

$$p = 101325 \times (1 - 2.25577 \times 10^{-5} h)^{5.25588} \tag{4.1}$$

where 101325 is the normal pressure at sea level in Pa, h and p are in meters and pascal respectively.
The BME 280 can be used in both SPI or I$^2$C communications types (see chapter 3 section 3.4.3) and thanks to the 3.3V internal regulator, it can be used even with a 5V logic e.g. Arduino Uno board. In the ESP8266 case the voltage regulator is useless because the chip already uses a 3.3V logic. To increase the environment's data

collection and also helping the estimation process[2], a photoresistor is also connected to the sole analogue input pin of the board. A Photoresistor or LDR (Light Dependent Resistor), is a light-controlled variable resistor. This component is usually very resistive (in the order of megaohms) when placed in the dark. However, when it is illuminated, its resistance decreases and may drop as low as a few hundreds of ohms, depending on the light's intensity. Photoresistors may require few milliseconds or more to fully respond to changes in light intensity, and their sensitivity and resistance range may vary from one device to another. Usually photoresistors are used in light/dark-activated switching circuits, in a circuit with a capacitor to avoid fluctuations of the output signal. In our circuit, to be measured by the analogue input pin, the photoresistor is included in a voltage divider with a comparable resistor (typically a 10kΩ).
In figure 4.2 is represented a simple circuit for the photoresistor along with its dimension and characteristics. As part of a voltage divider, the output voltage of the circuit follow

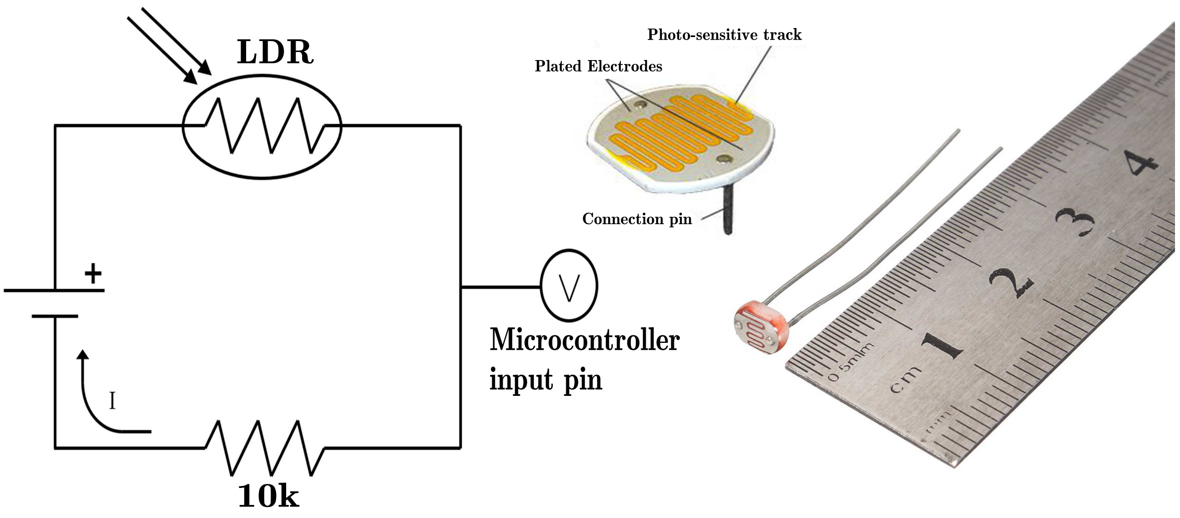


*Figure 4.2: A simple LDR circuit, its structure and dimensions.*

the equation:

$$V_{out} = \frac{R_{ph}}{R_{ph} \times R_1} \times V_{in} \tag{4.2}$$

As the intensity of the light increases, the resistance of the photoresistor decreases, so $V_{out}$ gets smaller[3] as more light hits the device. On the contrary as the intensity of the light decreases, the resistance of the photoresistor increase so $V_{out}$ gets bigger.
In the final project also a "send now" button is implemented. It works as an override

[2]Brightness can be correlated with the occupancy of the room e.g. when the lights are off and in the outside is dark, probably the room is empty.

[3]Remember the Ohm's law: V=R×I, where R is the resistance of the resistor expressed in ohms (Ω), V the voltage in volts (V), and I is the current in amperes (A).

control that forces the chip to send the actual people estimation and environmental data, ignoring the send-frequency selected by the user via Node-RED dashboard's user interface[4].

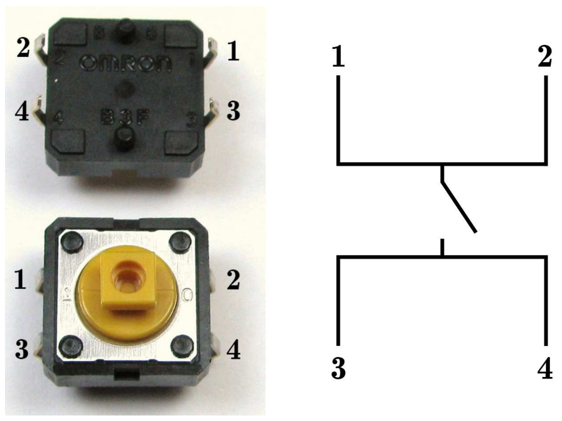
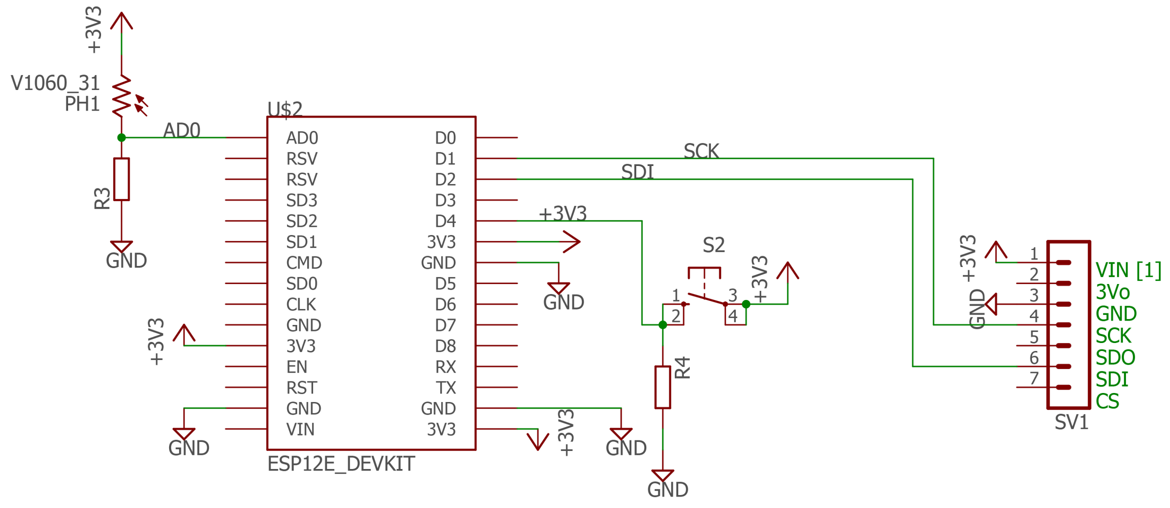


*Figure 4.3: Photo and circuit of a simple button.*

The working principle of a button is simple: when pushed, it makes a connection between legs 1-2 and 3-4 (see figure 4.3 for references) previously disconnected.
In our circuit (4.6) the pin is connected to port 1 of the button, while port 2 is connected to ground through a 10kΩ pull-down resistor. The port 4 of the button is connected to 3.3V. When the button is open (unpressed) there is no connection between 1-2 and 3-4 legs, so the pin is connected directly to the ground through the 10kΩ resistor. On button press, it makes a connection between the above nominated legs, connecting the 3.3V to the pin and pull-down resistor, making a voltage divider. By setting the purple pin in figure 4.6 as an input pin, and making a constant reading of that, the micro-controller can check the status of the input. In the case of open button the input will be LOW, because the pin will be at 0V. In the close button status, the input will be HIGH, because of the higher voltage.
In the next figure (4.4) the schematic of the circuit is shown. The schematic is a blueprint of the circuit, and must include all the information necessary so that everyone reading it can figure out what parts to buy and how to assemble the parts. For creating our schematic we used an electronic computer-aided design tool (CAD) named EAGLE, from Autodesk. One of the advantages of using a CAD software is that it provides two layouts: schematic and printed circuit board (PCB). In this way it's easier and faster to create the PCB layout from the schematic.
After the schematic, the next step is to make a prototype of the circuit, making sure that everything works well. The most common tool used for the prototype's assembly

[4]In the final project the user can select the time frequency at which receive people estimation and environment's updates. More details about this in sub-section 4.1.3 and section **??**



+3V3
V1060_31
PH1
AD0
R3
GND
U$2
AD0 RSV RSV SD3 SD2 SD1 CMD SD0 CLK GND 3V3 EN RST GND VIN
D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
ESP12E_DEVKIT
SCK
SDI
S2
R4
SV1
VIN [1]
3Vo
GND
SCK
SDO
SDI
CS

*Figure 4.4: The circuit's schematic created with Eagle CAD*

is a modular breadboard. A breadboard acts as a temporary assembly board on which all the electrical parts (sensors, resistors, microcontrollers etc.) are placed and joined together by wires or by the built-in conductive pathways underneath the surface of the board (figure 4.5).

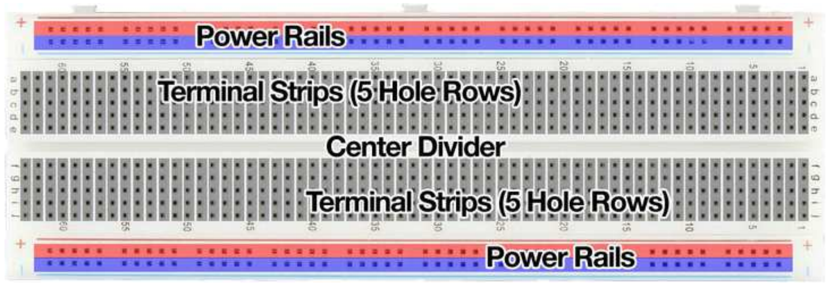


*Figure 4.5: Breadboard's inner connections.*

When a wire or component's pin is inserted into one of this sockets, it's held by two metal extremities. Because of their connections, the upper and lower rows are typically reserved for power supply connections, while the sockets between the central gab region are reserved for the others components (sensors, MCUs, buttons etc.)
Once connected the sensors, resistors, button and microcontroller in the breadboard, we obtain the prototype circuit shown in figure 4.6.
Brief explanation of the connections:

- The BME 280 uses only four pins out of the seven pins of its PCB. The $V_{in}$ and GND are connected respectively to the 3v3 and GND pin of the NodeMCU board. Instead of using SPI, that requires 4 pins, the sensor is connected via

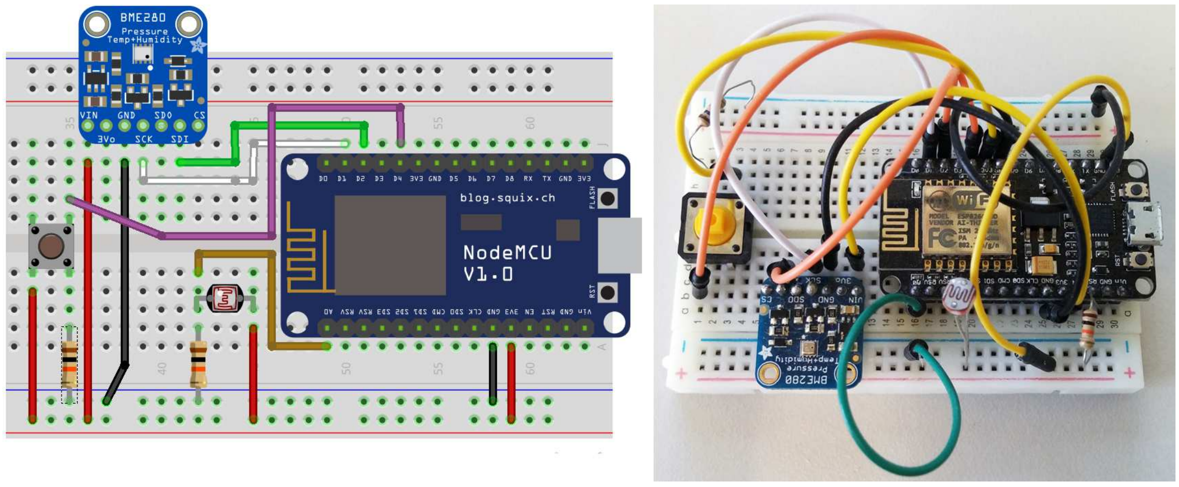

BME280
Pressure
Temp+Humidity
VIN GND SDO CS
3Vo SCK SDI
D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
blog.squix.ch
FLASH
NodeMCU
V1.0
RST
A0 RSV RSV SD3 SD2 SD1 CMD SD0 CLK GND 3V3 EN RST GND Vin

*Figure 4.6: Fritzing's prototype image, and real implementation photo*

$I^2C$[5] to the D1 and D2 pin of NodeMCU, enabled respectively to be used as SCL (Serial Control Line) and SDA (Serial Data Line).

- The Button, as you can see, is connected to the D4 pin of the board and, on the other side, to 3.3V and GND. When pressed, the D4 pin goes up to 3.3 putting the logical input value to "HIGH".

- The photoresistor is connected in parallel to the ground through 10kΩ resistor, and to the sole analogue pin of NodeMCU. The other pin is connected to the 3.3V input.

To simplify the circuit's reading, every connection to the ground is black, and to the source is red. Signals, instead, have different colours.

### 4.1.2 From the schematic to the Printed Circuit Board

Achieving portability through size reduction is an important part of developing a sensor. Passing from a proto-board (breadboard) to a PCB is justified by the size reduction (the breadboard is bigger than a custom-made board), packaging and reliability issues.
Creating the PCB layout and the actual PCB can be made in different ways: by hand with an etch-resistant pen, with a CAD software printing the layout and using photo-sensitive boards, and so on (see [14] for more informations).
For the first PCB of this project we use EAGLE CAD software, which we use to create the schematic, to generate the layout of the PCB out of the schematic. Once placed all

[5]See chapter 3 section 3.4.3 for details about SPI and $I^2C$.

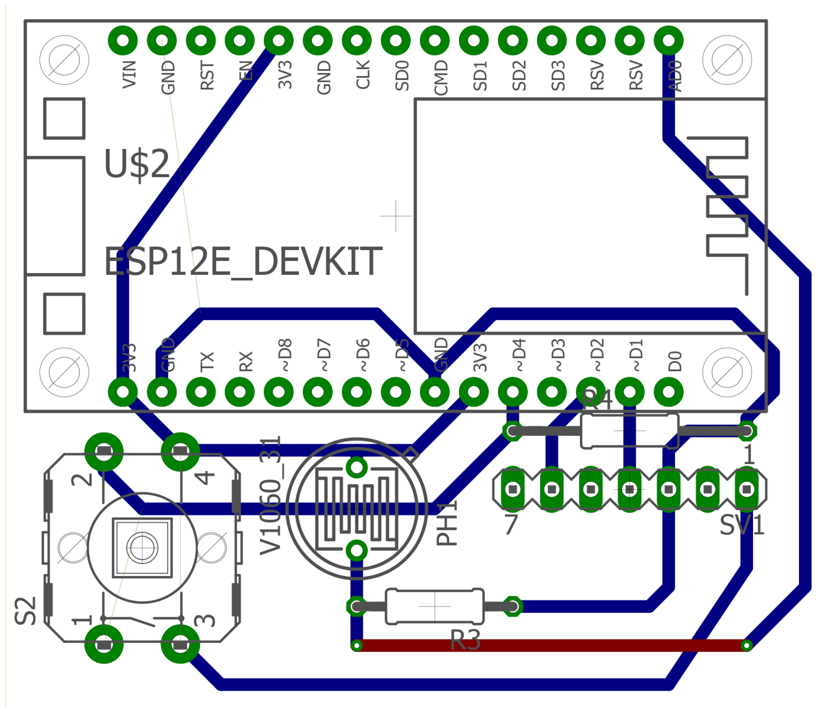


*Figure 4.7: PCB layout printed by EAGLE Cad software.*

the electronic elements in the layout, the routes among devices are created (manually, and not with the auto-route function), see figure 4.7.
Having at our disposition a 3D printer adapted to draw circuits, we used an etch-resistant pen and print the PCB layout onto an insulated board covered on one side with a very thin copper coating (figure 4.8).

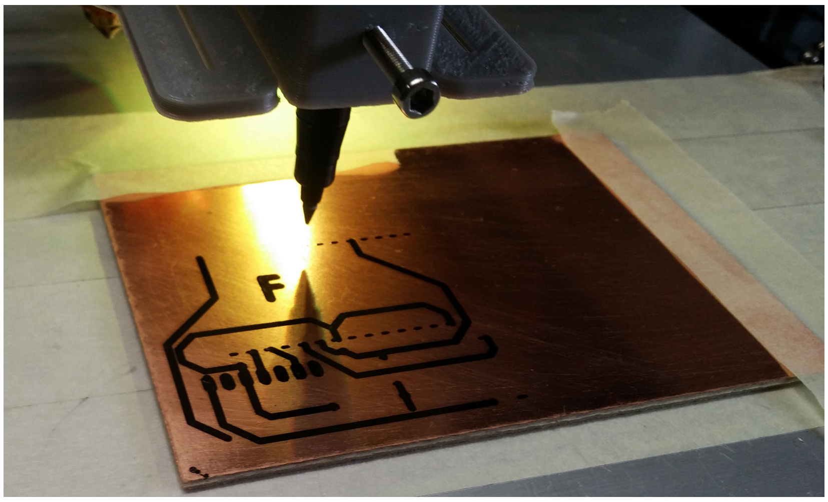


*Figure 4.8: A 3D printer adapted as a sort of CNC machine.*

After this procedure the board is placed in a tub of etching solvent (figure 4.9) and in a few minutes the copper dissolves away from the sections of the board that are not

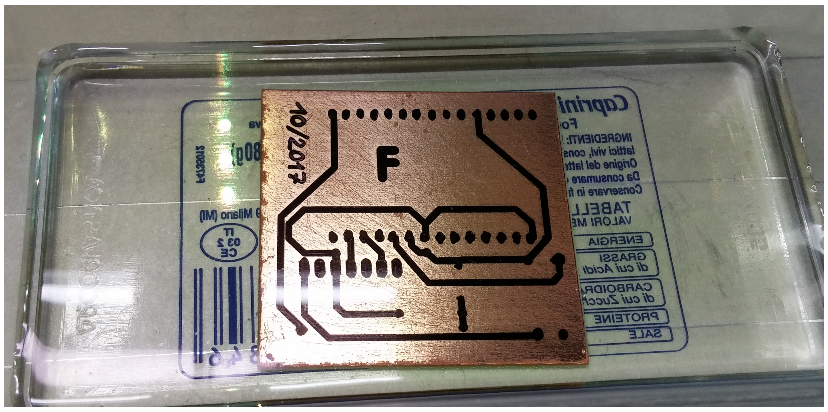


*Figure 4.9: The board immersed in a solution of hydrogen peroxide and muriatic acid*

cover by the marker.
After removing the marker's sign, the final PCB (4.10) is obtained and ready to be drill and solder with the sensors.

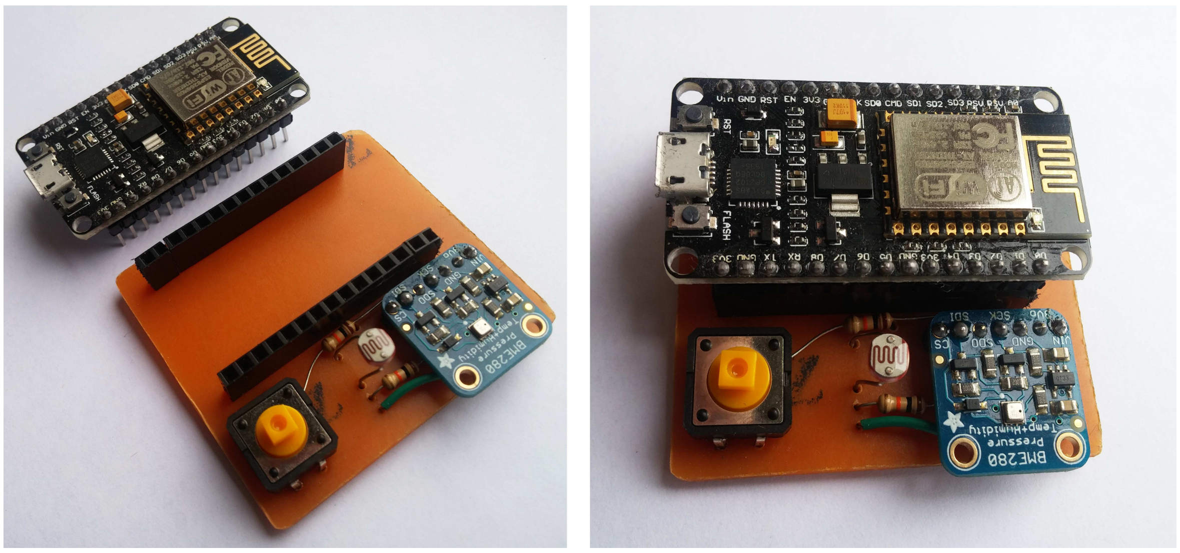
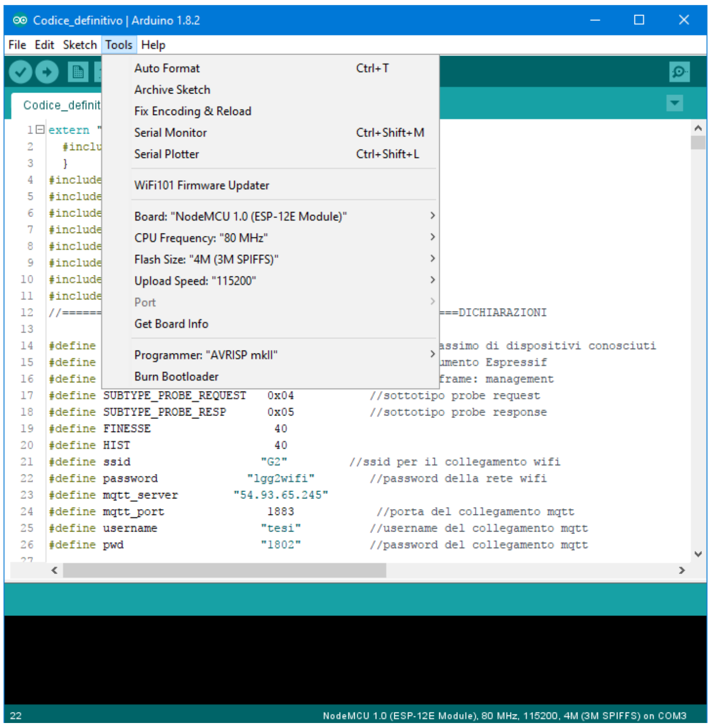


*Figure 4.10: Final circuit onto the PCB. The NodeMCU board is not soldered in order to be easy changed,upgrade or detach.*

In case of multiple copies of the circuit it's suggested the use of different techniques in order to actually make the PCB, or the use of specialized websites that provide the users great circuits at low costs.

### 4.1.3 The Program

A microcontroller without a program - a set of instructions - it's like a car without an engine. In this section the project's program is illustrated with focus on the "estimation function".
Every Arduino-like board can be programmed with the Arduino IDE software[6], that makes writing code and uploading it to the boards easier. The software run on Windows, Linux and Mac OS and it's written in java and based on the open-source software Processing. The ESP8266 breakout board comes with NodeMCU's Lua[16] interpreter, a firmware that permits the user to coding using Lua scripting language, a powerful, efficient and lightweight language that supports procedural programming, object-oriented programming, functional programming and data-driven programming. Despite its simplicity, for who knows C or C++ the use of Arduino IDE is convenient. To do so, the users must install the ESP8266 Board Package[7] and then select the correct board in the Tools menu.


Codice_definitivo | Arduino 1.8.2
File Edit Sketch Tools Help
Codice_definit
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "NodeMCU 1.0 (ESP-12E Module)"
CPU Frequency: "80 MHz"
Flash Size: "4M (3M SPIFFS)"
Upload Speed: "115200"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
22
NodeMCU 1.0 (ESP-12E Module), 80 MHz, 115200, 4M (3M SPIFFS) on COM3

```
extern "
  #inclu
  }
#include
#include
#include
#include
#include
#include
#include
#include
//======                                    ===DICHIARAZIONI

#define                                    assimo di dispositivi conosciuti
#define                                    umento Espressif
#define                                    frame: management
#define SUBTYPE_PROBE_REQUEST   0x04          //sottotipo probe request
#define SUBTYPE_PROBE_RESP      0x05          //sottotipo probe response
#define FINESSE                  40
#define HIST                     40
#define ssid                   "G2"        //ssid per il collegamento wifi
#define password             "lgg2wifi"       //password della rete wifi
#define mqtt_server        "54.93.65.245"
#define mqtt_port               1883           //porta del collegamento mqtt
#define username               "tesi"         //username del collegamento mqtt
#define pwd                    "1802"         //password del collegamento mqtt
```

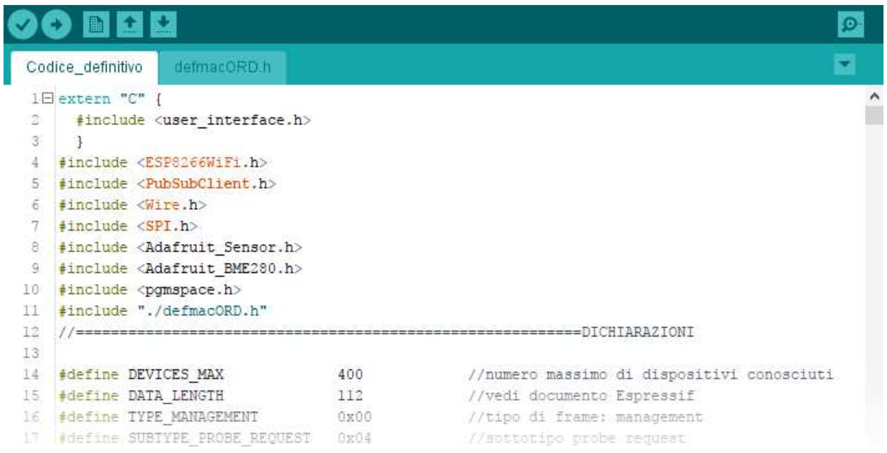
Figure 4.11: Arduino IDE interface in the board selection menu.

This procedure will write directly to the board's firmware, erasing the NodeMCU

[6]www.arduino.cc/en/main/software
[7]Packages can be installed by entering their URL into the *Additional Board Manager URLs* field in the Arduino IDE 1.6+ preferences. In the case of ESP8266 the added URL was: $http : //arduino.esp8266.com/stable/package_esp8266com_index.json$

firmware with its Lua interpreter.
Figure 4.11 shows the settings in the tool menu when the correct board (NodeMCU 1.0 in this case) is selected.
The first part of the program is the libraries' declaration. In this project we used the mqtt protocol, the BME280 sensor in $I^2C$ configuration, and the Wi-Fi sniffing and normal mode of the ESP8266. In the program are therefore included the respective libraries that are:

- **ESP8266WiFi.h**: the library containing all the functions to connect to an AP, check connections and more.
- **PubSubClient.h**: provides a client for doing simple publish/subscribe messages to a server that supports MQTT.
- **Wire.h**: It's the library that enables $I^2C$ connection.
- **Adafruit_Sensor.h**: this one and the sequent is the library with functions for enabling Adafruit's BME280 data capture.
- **Adafruit_BME280.h**: Same as the above one, providing specific functions for the BME like the "BME.begin()".
- **pgmspace.h**: this library enable/correct the use of the PROGMEM function in ESP8266 to save data into the program storage space instead of the dynamic memory.
- **user_interface.h**: this library add the promiscuous mode support in the Wi-Fi chip.



```
extern "C" {
  #include <user_interface.h>
  }
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <pgmspace.h>
#include "./defmacORD.h"
//===========================================================DICHIARAZIONI

#define DEVICES_MAX            400          //numero massimo di dispositivi conosciuti
#define DATA_LENGTH            112          //vedi documento Espressif
#define TYPE_MANAGEMENT        0x00         //tipo di frame: management
#define SUBTYPE_PROBE_REQUEST  0x04         //sottotipo probe request
```

Figure 4.12: The libraries included in the program in the Arduino IDE.

In the "include" part it is also added an *#include "./defmacORD.h"* that is not a library but a ".h" file containing the OUI - Organizationally Unique Identifier[5] a 24-bit (first 3 bytes of the MAC address) hexadecimal number that uniquely identifies the manufacturer of the wireless card. More about this file and its usage will be discussed later on.
After the libraries inclusion, the define and declaration part of global variables begins. Some constant variables, useful to understand the program, are: the MQTT topic's name; the broker's IP, port, username and password; the sensors' pins; the AP ssid and password; the range of powers considered. For this last one parameter a detailed explanation is necessary: every probe packet captured comes with a RSSI field that gives an idea of the distance between sensor and device. Choosing to filter the probes reception to packets beneath a certain power meant to limit the range of the probe capture process adapting it to the size of the room. With this is mind, the program divides the power into 40 values from -44dB to -120dB (not uniformly distributed, but packed between -70dB and -90dB), and calculate the number of devices between 0dB and the selected value, making a cumulative distribution function of the number of devices e.g. N_devices(-40dB) gives the number of devices from 0dB to -40dB, and N_devices(-60dB) gives the number of devices from 0dB to -60dB, not only between -40 and -60dB. As a matter of fact, more dense is the power range considered, more accurate will be the prediction on the number of people.
On the other hand, some non-constant variables are: the send frequency (changed on-demand by the user), the current model variables (4.1.3.2), state variables (like the state of the button or the internal LED), actual captured packet for each power step (as explained before) and past history (a registry that keeps memory of last 40 measurements and is shifted by any new measure input).
The flowchart in figure 4.13 shows a panoramic of the program operation. For the sake of clarity, the program can be divided into three main parts:

**Set-up** This part is the upper part of the flowchart, and runs only at the first start-up (or reboot) of the sensor. When just powered up, the ESP tries to connect to the AP[8] and retrieve data from the MQTT broker to check the last model's parameters used and set them in the program (more about the estimation function's parameters in 4.15 and its explanation). After retrieving (or not) the data, the program pass to the next step, putting the Wi-Fi chip in sniffing mode.

[8]The AP name and password are saved into the program, so every ESP must have the right AP memorized, or a set of them.

**Sniffing** This is the main part of the program, the part at which the ESP is for the 80-90% of time, and is the left one in the flowchart 4.13. After activate the sniffing mode, the ESP starts "listening" to other devices' requests 3.1.3.1 and saving their IDs (MAC addresses) and RSSI. This scanning stage is run on a single channel (the channel 1, one of the three non-overlapping channels of Wi-Fi, see 3.1.1). Although in [12] the best performance (in terms of number of captured probes) was obtained using multiple antennas at different channels, in our project we were not able to reproduce this method. We tried to implement a system that changes channel every fixed amount of time instead, but the measurements obtained shows that there were not important differences with the single channel implementation.

When a new probe arrives the microcontroller checks if the MAC has already been saved in the past 7 minutes[9] or it's a new unknown device. In the first case the program updates the RSSI field of the saved MAC and update the number of devices for each power range. In the latter case there is no record about the device yet, so the program creates a new record with 4 fields: MAC address, RSSI, last time seen and a flag named *Rndm* with value 1 or 0 depending of the type of MAC address found (see 3.1.3.1). To recognize random versus non-random IDs the *#include "./defmacORD.h"* comes into play: this file contain all the registered OUIs assigned by the IEEE. The program checks[10] if the first six digit of the captured MAC address are part of this list, and as a results puts the *Rndm* flag equal to 1, if the ID is not present in the OUI's list, and to 0 if it is. After the new record is added, the program updates the number of devices found with respect to the power range.

There are two "N_devices" variables depending on the *Rndm* flag: the first updated when the device have a non-randomized MAC address, and the second one updated when the device have a random MAC address.

These operations are made for each probe packet captured.

**Report** This is the right part of 4.13 and is activated on-demand by pressing the button, or setting the "send timer" via web interface (4.33). After one of those events, the ESP notify the entrance in this part of the program lighting up its internal LED; the promiscuous mode is stopped, and the chip connects to the AP. After the connection to the Access Point and the internet, the ESP connects to the MQTT

[9]After a period of 7 minutes, if there are no probe captured for a certain MAC address, this is erased from the list presuming that the device is no longer present in that area

[10]A *Binary search algorithm* is implemented in order to reduce the searching time with respect to a normal search (with a gain of at least 30ms on the process' time).

broker to retrieve the "send frequency" decided by the user, the real number of people and a flag called "PowerLock". This last flag decide the sequent tasks because it tells if the "number of people" value retrieved is still valid or expired (it expires after 10 minutes from its last modification).
If the PowerLock value is 1, the real number of people retrieved is not valid. In this case the MCU has to esteem the actual number of people using its previous model's parameters (4.1.3.2), retrieve the sensors' data (brightness, temperature, humidity, altitude and pressure) and send all these data to the right MQTT topics.
If the PowerLock value is 0, the real number of people retrieved is valid. In this case the model's parameters have to be calculated to minimize the Root Mean Square Error function, using the real number of people retrieved. After this operation, as above, the ESP retrieve the sensors' data and send them along with the people estimation and, in addiction, the devices' count for every power range, either the random and non-random count.
The last tasks of this part of the program are in common: the internal LED is switched off, the timer that starts this part of the program when the "send frequency" expires is reset and the Wi-Fi module is switched on sniffing mode to re-start the part 2 of the program.

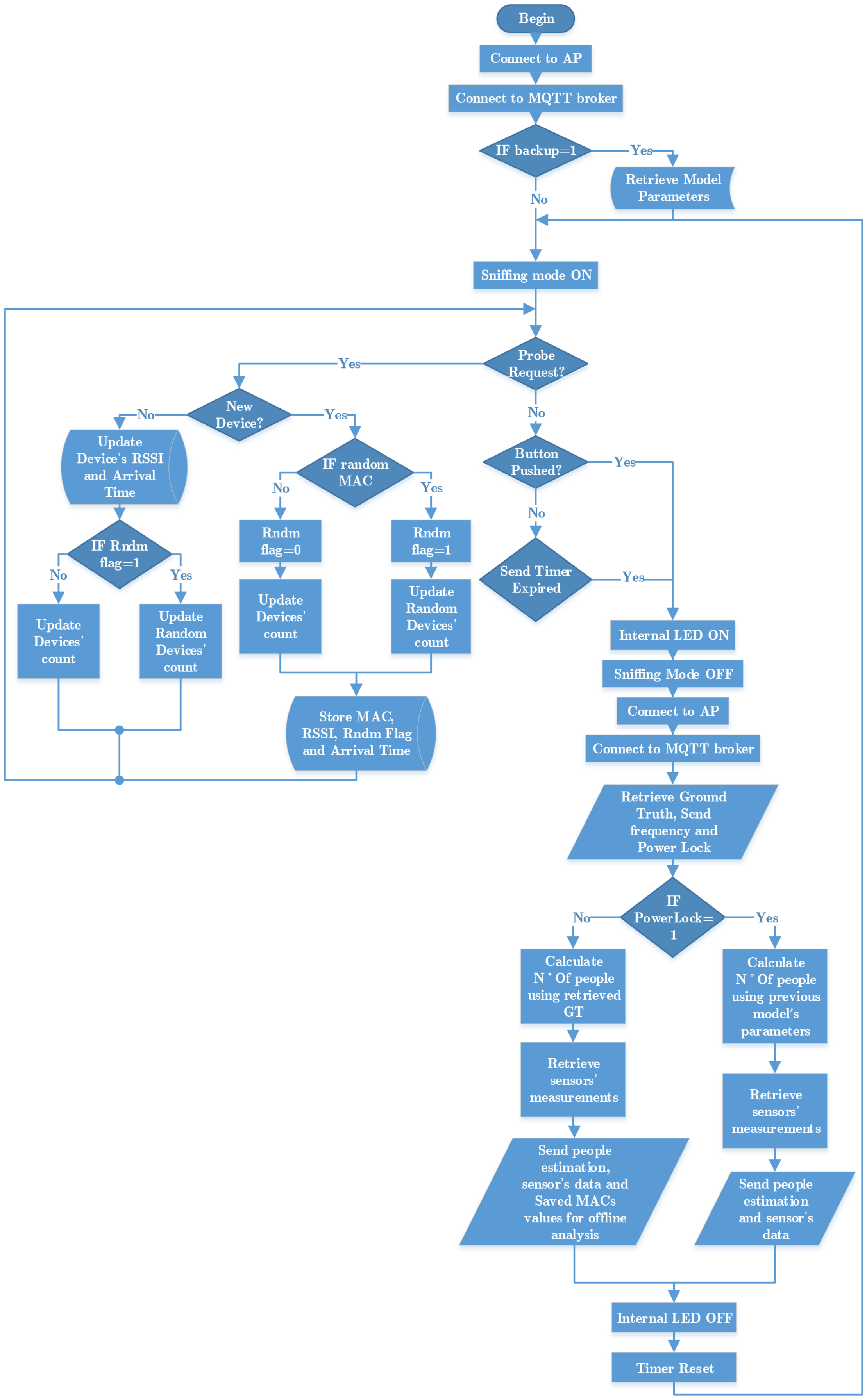

Begin
Connect to AP
Connect to MQTT broker
IF backup=1
Yes
Retrieve Model Parameters
No
Sniffing mode ON
Probe Request?
Yes
New Device?
No
Update Device's RSSI and Arrival Time
IF Rndm flag=1
No
Update Devices' count
Yes
Update Random Devices' count
Yes
IF random MAC
No
Rndm flag=0
Update Devices' count
Yes
Rndm flag=1
Update Random Devices' count
Store MAC, RSSI, Rndm Flag and Arrival Time
No
Button Pushed?
Yes
No
Send Timer Expired
Yes
Internal LED ON
Sniffing Mode OFF
Connect to AP
Connect to MQTT broker
Retrieve Ground Truth, Send frequency and Power Lock
IF PowerLock=1
No
Calculate N° Of people using retrieved GT
Retrieve sensors' measurements
Send people estimation, sensor's data and Saved MACs values for offline analysis
Yes
Calculate N° Of people using previous model's parameters
Retrieve sensors' measurements
Send people estimation and sensor's data
Internal LED OFF
Timer Reset

*Figure 4.13: The program's flowchart.*

In order to simplify the flowchart 4.13 a function, together with the estimation model, were omitted.
This function is the one that remove a device informations after seven minutes without receiving its probe requests. This is a simple function that is necessary in order to keeping track of the people, with devices, that went away from the room. This period of time has been chosen given [12], where in page 5 is shown a series of histograms about the time between bursts of probe requests for different devices. Choosing an interval of 420 seconds (7 minutes) seems to be a reasonable time interval after which a devices can be supposed gone. Of course is not 100% possible to guarantee that after 7 minutes of silence a device is gone. Tests on some devices shown that not all of them sent probes when the screen is turned off, depending on the OS installed and manufacturer personalizations. Flowchart 4.14 briefly illustrate how this function works and where, in the the first flowchart, is placed.

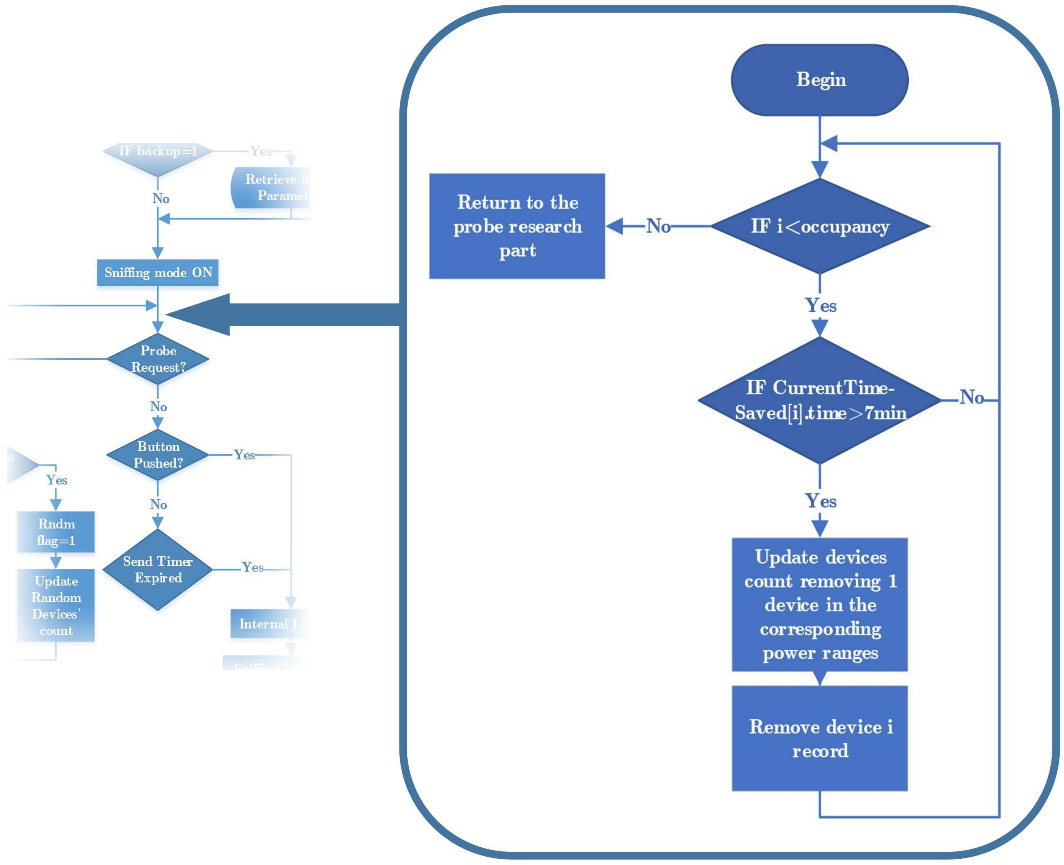

Begin
IF i<occupancy
No
Return to the probe research part
Yes
IF CurrentTime-Saved[i].time>7min
No
Yes
Update devices count removing 1 device in the corresponding power ranges
Remove device i record
IF backup=1
Yes
No
Sniffing mode ON
Probe Request?
No
Button Pushed?
Yes
No
Send Timer Expired
Yes
Rndm flag=1
Update Random Devices' count
Internal

Figure 4.14: The removing function.

#### 4.1.3.1 Power-safe mode

It's useful to keep sending data during night? As a response to this question, we decide to optimize the power consumption of the chip, enabling a so-called "power-safe mode". This part is omitted in 4.13, in order to simplify its comprehension. Since we have provided our board with a light sensor, we decide to make a good use of it: during the nights, when the lights are switched off and the outside environment light is low, the chip starts to lower its send frequency. During its "Report" part, the program reads the brightness value given by the photoresistor. When this value goes under a certain threshold for more that one consecutive send time, the chip supposed that the lights are off. Intepolating this data with the estimated number of devices (supposed to be zero during nights), the chip stops its sniffing mode, power off the modem, and reduce the send frequency (which requires the connection to the AP, so the powering on of the Wi-Fi modem). When, during another measurements, the brightness value rise, the chip returns to its normal operation mode, using the user's imposed send frequency, and turning on the sniffing mode.

#### 4.1.3.2 The Estimation Model

The very core of this project is the algorithm to get the correct number of person in a room knowing the probe requests' power distribution. A correct and efficient algorithm bring the ESP to make the correct estimation without too much computing power, since in a MCU calculations in floating-point variables are a real problem.
In this project we implemented a simple but effective algorithm that, taking into account the last 40 measurements, tries to minimize the error between the estimation and the reality. The history is made out by 40 elements, due to space limitation into the microcontroller: a higher value will saturate the internal memory during its runtime, because implies larger vectors, records and longer calculations.
Let's see in details the model defining some parameters. Let:

$N_T(t)$ be the sole model's input parameter, indicating the real number of persons in the room at time t. This input is retrieved from the MQTT server before sending the estimation and ambient measurements (see the right part of 4.13).

$N_V^\theta(t)$ be the number of devices seen by the Wi-Fi chip at time t and power $\geq \theta$. The maximum amount of event t saved is 40, and the power ranges are also 40. So this variable is a table having the power range as columns, and events as rows.

$N_R^\theta(t)$ be the number of random MACs seen by the ESP. This is the equivalent, for the random MACs, as the table before where the columns are the power ranges, and the rows the events.

$\alpha$ be a correction factor for $N_V^\theta(t)$. Since one person can use more than one device with Wi-Fi enabled (smartphone, notebook, smartwatch, tablet and so on) or don't have any Wi-Fi device at all (or with Wi-Fi disabled), this factor corrects the occupancy's over or under estimation. Its range varies between 0.1 (1 person every 10 different non-random MACs received) and 2 (2 person per probe requests e.g. half students has one device with Wi-Fi enabled and the other half don't).

$\beta$ be the counterpart of the $\alpha$ factor for $N_R^\theta(t)$, the number of random MACs received at time t. Supposing that a devices sends probes with 3 different MACs, the $\beta$ value will be 0.3. This value varies from 0 (it completely ignores random MACs) to 2, and must always be less than the $\alpha$ value. This last imposition is based on the assumption that the number of random MACs changes more frequently, is more unpredictable. For this reason is more "risky" to make an estimation based mainly on the random MACs number.

$n$ be the number of inputs considered in the calculation. At first start-up the value will be 0 and will increase with every new input until 40. Once reached this maximum value, every new input will replace the "oldest" one and so on, leaving the total input considered to 40.

The formula behind the estimation is:

$$\min_{\alpha,\beta,\theta} \sqrt{\frac{1}{n}\sum_{t=1}^{n}(N_T(t) - \alpha N_V^\theta(t) - \beta N_R^\theta(t))^2} \tag{4.3}$$

with $\theta$=1,2,...,40; $\alpha$=0.1,0.2,...,2; $\beta$=0,0.1,...,2 and $\alpha > \beta$

In conclusion, at every new input, the model calculates the values of the parameters $\alpha$,$\beta$ and $\theta$ to minimize the overall error. In this way, we create a dependence between the new and past parameters of the model.
Let's call $\alpha^*$, $\beta^*$ and $\theta^*$ the values that minimize the above equation. Therefore the estimation at t+1 instant is calculated as:

$$\hat{N}(t+1) = \alpha^* N_V^\theta(t+1) + \beta^* N_R^\theta(t+1) \tag{4.4}$$

where the result is rounded to the nearest integer (we cannot say that there are 3.4 person in the room, we have to round the estimated value). The results of the appli-

cation of this model can be seen in chapter 5.
It's now more clear that, due to the low computing power and storing capabilities (already under pressure with the list of OUIs), limit the number of inputs taken into account in the formula was a necessity, not an option. To make more clear how much hard this cycles are, we calculate the time interval of this part of the program, and the sniffing part:
the program requires approximately 1ms of time to capture, analyse, and save one probe request.
To calculate the correct parameters having already 40 inputs, the program takes almost 3.5 seconds. It's clear now that the biggest problem was, on a chip like this, avoiding the automatic reboot due to overload (see the watchdog timer in 3.4.2.1). To solve this problem we use a function that, during every cycle, "kick" the watchdog timer, in order to avoid its expiration.
The next flowchart (4.15) illustrates the application of the 4.3 formula in the program. To simplify the 4.13 this model part was omitted, and the next figure shows in which part of the first flowchart this calculation takes part.

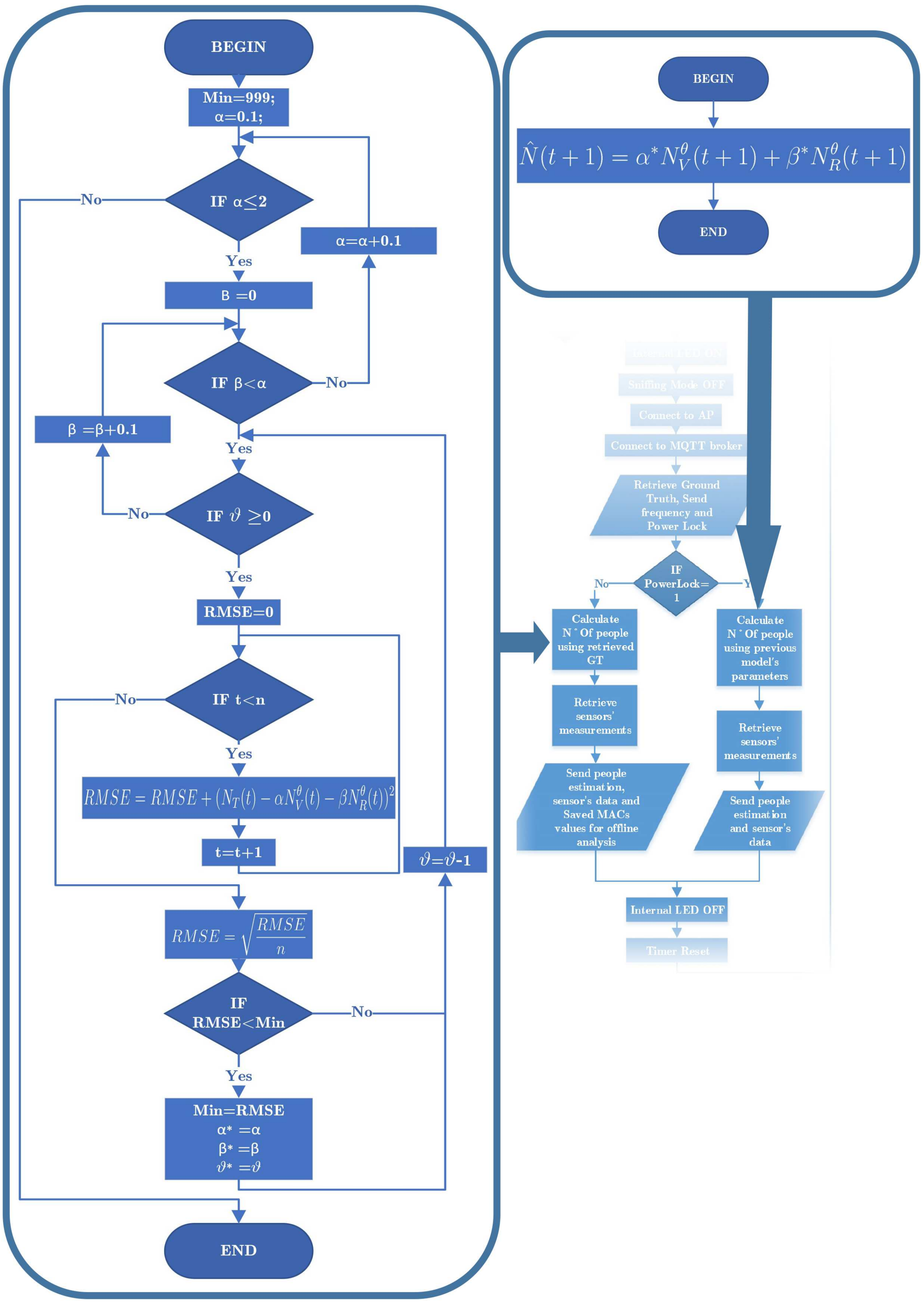

BEGIN
Min=999; α=0.1;
IF α≤2
No
Yes
α=α+0.1
Β =0
IF β<α
No
Yes
β =β+0.1
IF ϑ ≥0
No
Yes
RMSE=0
IF t<n
No
Yes
$RMSE = RMSE + (N_T(t) - \alpha N_V^{\theta}(t) - \beta N_R^{\theta}(t))^2$
t=t+1
ϑ=ϑ-1
$RMSE = \sqrt{\frac{RMSE}{n}}$
IF RMSE<Min
No
Yes
Min=RMSE
α* =α
β* =β
ϑ* =ϑ
END
BEGIN
$\hat{N}(t+1) = \alpha^* N_V^{\theta}(t+1) + \beta^* N_R^{\theta}(t+1)$
END
Internal LED ON
Sniffing Mode OFF
Connect to AP
Connect to MQTT broker
Retrieve Ground Truth, Send frequency and Power Lock
IF PowerLock= 1
No
Y
Calculate N ° Of people using retrieved GT
Calculate N ° Of people using previous model's parameters
Retrieve sensors' measurements
Retrieve sensors' measurements
Send people estimation, sensor's data and Saved MACs values for offline analysis
Send people estimation and sensor's data
Internal LED OFF
Timer Reset

*Figure 4.15: The estimation model's function is a "brute force" research of the correct coefficients and power value.*

**Model B**

The final model discussed until now was not the first model implemented in the project. At the very beginning, one of the first codes implements a different type of model: the MAC addresses recognized as random were discarded. This model is the equivalent of putting the $\beta$ variable to a fixed zero value. Although, since the MAC randomization is a growing phenomena (with newest mobile and PC OSs releases), the final model is more "enduring" in the long period. In the next future, the random MAC probes will probably overtake the non-random values, and for this reason the $\beta$ value is fundamental for our purposes.
The results of this "Model B" are shown in the next chapter, along with the results of the final model.

## 4.2 Server-Side Implementation

The server part comprehend the MQTT broker and the Node-RED environment, extensions and flows. This section is therefore divided into four main parts:

1. The server's choice
2. MQTT broker installation and configuration
3. Node-RED, installation, configuration, expansion
4. The Node-RED flow - the server part's core

### 4.2.1 The Server

Run an MQTT broker and Node-RED client is not a very difficult task (in terms of computing power). With this in mind our first choice was to implement the server-part on a Raspberry Pi 3, a single-board low cost computer with: Broadcom quad core 1.2Ghz 64-bit CPU, 1GB of RAM, Wi-Fi, Bluetooth and Ethernet connections, 4-USB 2.0 ports, 40 GPIO pins, an HDMI port and, more important, a consumed power between 1.5W, when in idle state, and 6.7W under heavy usage.

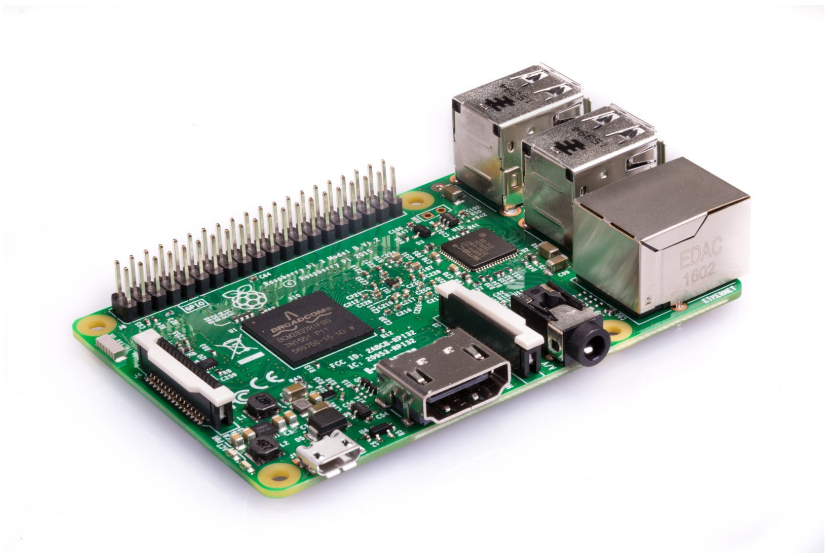


*Figure 4.16: The Raspberry Pi 3 Model B single-board computer*

The only lack of this implementation a fixed IP address of the network connection. The Raspberry Pi was connected to the ISP[11] network by the ethernet cable through a

[11]ISP stands for Internet Service Provider, an organization that provides access to the internet as long as other services like (but not always) web hosting, domain name registration, fixed telephone line access, media streaming, mobile access etc.

modem-router. In the local network the RPI had a fixed IP, given by the router. The ISP although, doesn't offer a fixed IP option from home users, therefore the IP address with which the internet "sees" the server (RPI in this case) was changing every time. For a server to be reachable from the outside a fixed IP is needed.
At the beginning the solution was to use a DDNS[12] service, to make the MQTT server always available from the outside with a fixed name. The DDNS is a technology that allows users to give their home network PC or server a permanent address on the internet. With DDNS although internet service providers change the user's IP on a regular basis, he can keep his domain name pointed to the current IP address of the home server/PC.
The implementation works like this:

1. The user chooses a service (in our case www.dynu.com) and registers to it.

2. A thir level domain name (usually free) or a top level domain name can be chose. In our case we chose a third level domain name eg. example.dynu.com. After this choice the domain name is created and updated with the current IP address used.

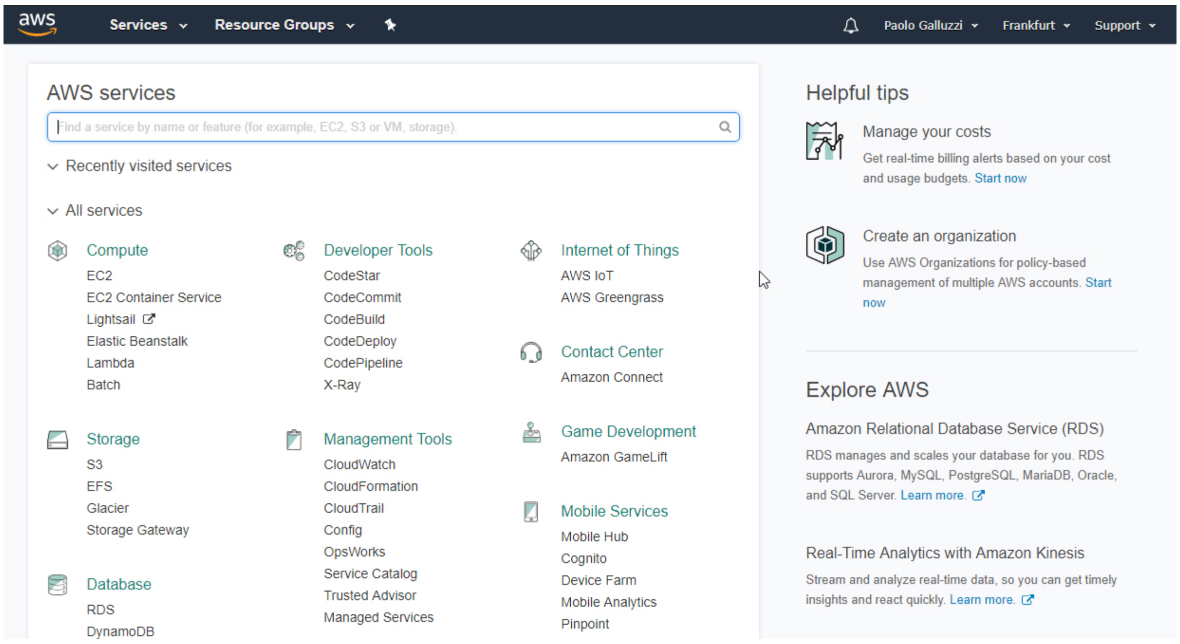
3. Install the client software, a daemon that keeps running in background even when no one is logged into the system.

4. Configure the software by insert the username, password and the update rate. The program will now on update automatically the IP pointed from the chosen DNS at the chosen update rate.

This method works well for the MQTT server. Unfortunately once Node-RED was installed it was difficult to be reachable from the outside. Since the aim of the project was to have a website reachable with ease by every student in and out of the campus, this first choice was not suitable for the purpose.
Due to poor connection speed and the necessity to have a fixed IP address for Node-RED's dashboard accessibility (see next sections for details about Node-RED dashboard), the project was moved on an Amazon EC2 virtual server through a free subscription to the Amazon Web Services. Amazon's Elastic Compute Cloud is one of the central part of Amazon's cloud computing platform: it allows user to rent virtual computers on which they can run their applications. Although in the near future the idea is to bring the system on the campus' servers, in order to get rid of the dependence on other external companies, and avoid fees that will be applied after the 12-Month

[12]Dynamic Domain Name System

Free Tier account's expiration.
The AWS console is shown in figure 4.17 where in the compute part the EC2 services can be seen.



AWS services

Find a service by name or feature (for example, EC2, S3 or VM, storage).

Recently visited services

All services

| Compute | Developer Tools | Internet of Things |
|---|---|---|
| EC2 | CodeStar | AWS IoT |
| EC2 Container Service | CodeCommit | AWS Greengrass |
| Lightsail | CodeBuild | |
| Elastic Beanstalk | CodeDeploy | **Contact Center** |
| Lambda | CodePipeline | Amazon Connect |
| Batch | X-Ray | |
| **Storage** | **Management Tools** | **Game Development** |
| S3 | CloudWatch | Amazon GameLift |
| EFS | CloudFormation | |
| Glacier | CloudTrail | **Mobile Services** |
| Storage Gateway | Config | Mobile Hub |
| | OpsWorks | Cognito |
| **Database** | Service Catalog | Device Farm |
| RDS | Trusted Advisor | Mobile Analytics |
| DynamoDB | Managed Services | Pinpoint |

Helpful tips

Manage your costs
Get real-time billing alerts based on your cost and usage budgets. Start now

Create an organization
Use AWS Organizations for policy-based management of multiple AWS accounts. Start now

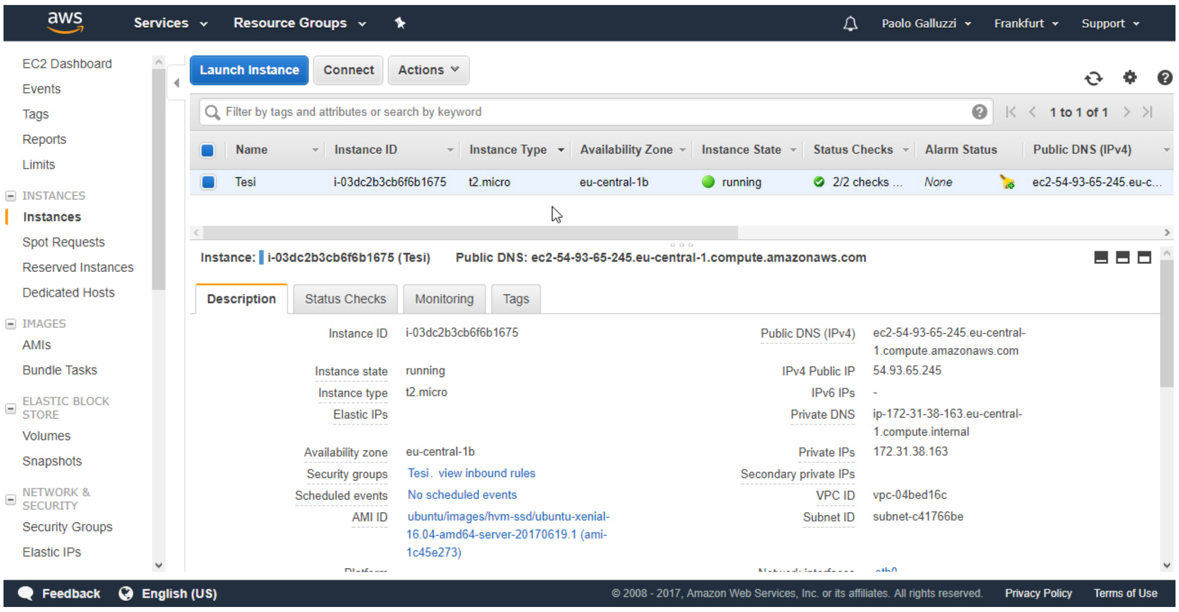
Explore AWS

Amazon Relational Database Service (RDS)
RDS manages and scales your database for you. RDS supports Aurora, MySQL, PostgreSQL, MariaDB, Oracle, and SQL Server. Learn more.

Real-Time Analytics with Amazon Kinesis
Stream and analyze real-time data, so you can get timely insights and react quickly. Learn more.

*Figure 4.17: Amazon Web Services console's main page*

In 4.18 the Instances (EC2) page can be seen. In this case the instance running is the one for this project, where you can see the details and IPv4 public IP.



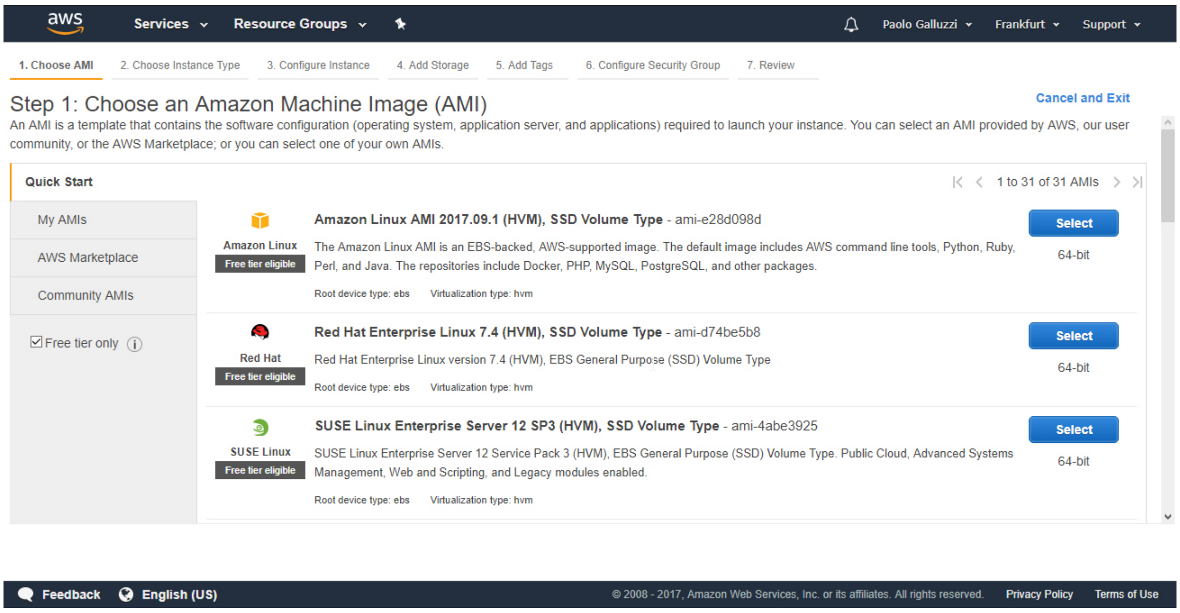
| Name | Instance ID | Instance Type | Availability Zone | Instance State | Status Checks | Alarm Status | Public DNS (IPv4) |
|---|---|---|---|---|---|---|---|
| Tesi | i-03dc2b3cb6f6b1675 | t2.micro | eu-central-1b | ● running | ✓ 2/2 checks ... | None | ec2-54-93-65-245.eu-c... |

Instance: i-03dc2b3cb6f6b1675 (Tesi)    Public DNS: ec2-54-93-65-245.eu-central-1.compute.amazonaws.com

| | | | |
|---|---|---|---|
| Instance ID | i-03dc2b3cb6f6b1675 | Public DNS (IPv4) | ec2-54-93-65-245.eu-central-1.compute.amazonaws.com |
| Instance state | running | IPv4 Public IP | 54.93.65.245 |
| Instance type | t2.micro | IPv6 IPs | - |
| Elastic IPs | | Private DNS | ip-172-31-38-163.eu-central-1.compute.internal |
| Availability zone | eu-central-1b | Private IPs | 172.31.38.163 |
| Security groups | Tesi. view inbound rules | Secondary private IPs | |
| Scheduled events | No scheduled events | VPC ID | vpc-04bed16c |
| AMI ID | ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-20170619.1 (ami-1c45e273) | Subnet ID | subnet-c41766be |

*Figure 4.18: Instances' page, where running instances are managed and new ones created.*

As seen in the next figures, when you create an instance with an AWS free tier, you can choose between several AMIs (Amazon Machine Images, 4.19) of different OSs. Our is running on Ubuntu Server 16.04.



Step 1: Choose an Amazon Machine Image (AMI)

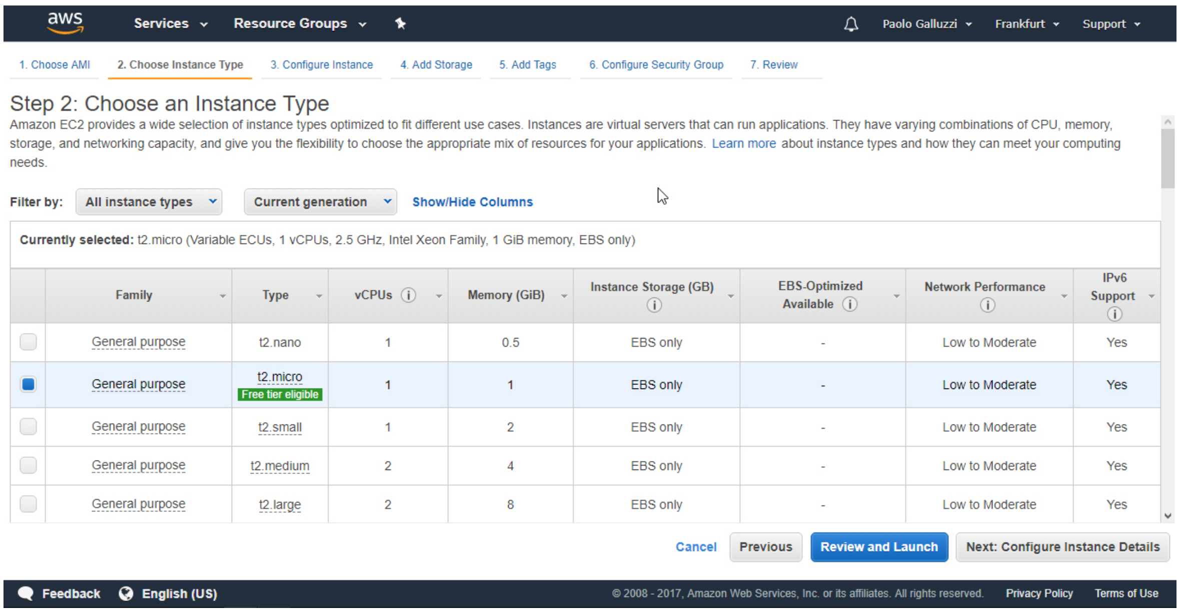
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

| | | |
|---|---|---|
| Amazon Linux<br>Free tier eligible | **Amazon Linux AMI 2017.09.1 (HVM), SSD Volume Type** - ami-e28d098d<br>The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.<br>Root device type: ebs Virtualization type: hvm | Select<br>64-bit |
| Red Hat<br>Free tier eligible | **Red Hat Enterprise Linux 7.4 (HVM), SSD Volume Type** - ami-d74be5b8<br>Red Hat Enterprise Linux version 7.4 (HVM), EBS General Purpose (SSD) Volume Type<br>Root device type: ebs Virtualization type: hvm | Select<br>64-bit |
| SUSE Linux<br>Free tier eligible | **SUSE Linux Enterprise Server 12 SP3 (HVM), SSD Volume Type** - ami-4abe3925<br>SUSE Linux Enterprise Server 12 Service Pack 3 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.<br>Root device type: ebs Virtualization type: hvm | Select<br>64-bit |

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.

*Figure 4.19: AMIs selection page.*

In the sequent page you can choose between several server's hardware configuration.



Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

**Currently selected:** t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

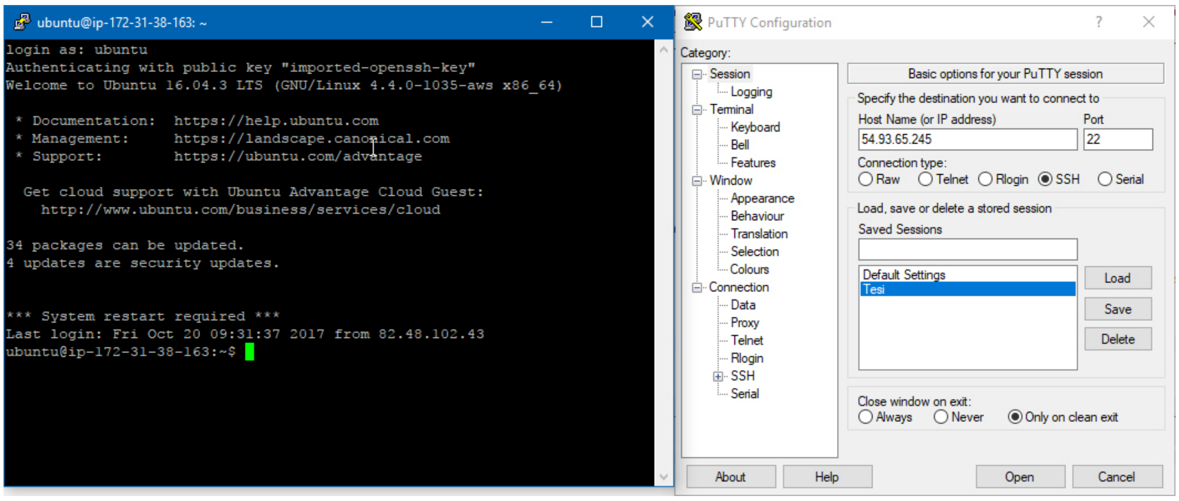
| | Family | Type | vCPUs | Memory (GiB) | Instance Storage (GB) | EBS-Optimized Available | Network Performance | IPv6 Support |
|---|---|---|---|---|---|---|---|---|
| ☐ | General purpose | t2.nano | 1 | 0.5 | EBS only | - | Low to Moderate | Yes |
| ■ | General purpose | t2.micro<br>Free tier eligible | 1 | 1 | EBS only | - | Low to Moderate | Yes |
| ☐ | General purpose | t2.small | 1 | 2 | EBS only | - | Low to Moderate | Yes |
| ☐ | General purpose | t2.medium | 2 | 4 | EBS only | - | Low to Moderate | Yes |
| ☐ | General purpose | t2.large | 2 | 8 | EBS only | - | Low to Moderate | Yes |

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.

*Figure 4.20: The instance's type selection. Is the selection of the hardware of th virtual machine.*

In our case, for the free tier, the t2.micro was the only available option. It is based on 1 virtual CPU@2.5GHz clock speed, from the Intel Xenon family, and 1GB of RAM. The storage memory chosen is of 40GB, that for our purposes is fully enough. After

this procedures, the important part is to set the right "Security Group", the set of rules that manage what connections the server can receive or make. In our case we have to open the SSH port 22 - the Secure Shell protocol that allow the user to establish a ciphered remote session through command line, useful to manage the server, install the programs and so on. The MQTT port 1883, to allow our sensor to connect through MQTT to the server and send data. At last the Node-RED port 1880, in order to access to the flows from the outside. Of course the MQTT and Node-RED port can be changed, but we used the default ones. Since Node-RED flows can be viewed by any user who knows the public IP and port, in the configuration phase a login page was created (more details later on).
Once the creation of the server is finished a key is given. This key must be used in order to connect through SSH to the server, (see figure 4.21) with which it is possible to install the programs needed.


ubuntu@ip-172-31-38-163: ~

```
login as: ubuntu
Authenticating with public key "imported-openssh-key"
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

34 packages can be updated.
4 updates are security updates.


*** System restart required ***
Last login: Fri Oct 20 09:31:37 2017 from 82.48.102.43
ubuntu@ip-172-31-38-163:~$
```

PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 54.93.65.245

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions: Default Settings, Tesi

Close window on exit: Always Never [✓ Only on clean exit]

*Figure 4.21: The SSH connection with PuTTY client on Windows.*

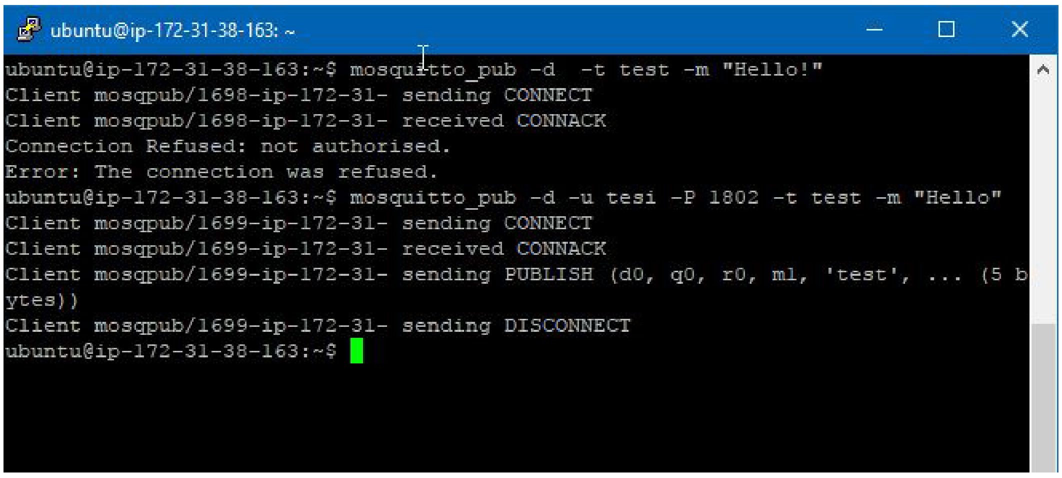
### 4.2.2 The MQTT Broker - Installation and set-up

Once the server is running, the various installations processes can begin. The first thing to do is to choose the MQTT broker. In our project we've installed the open-source message broker Mosquitto[13] that implements the MQTT protocol version 3.1. Since our server was based on Ubuntu 16.04 that already has a version of Mosquitto in its default software's repository, to install the broker only an "apt-get" line was necessary:

*sudo apt-get install mosquitto mosquitto-clients*

[13]https://mosquitto.org/

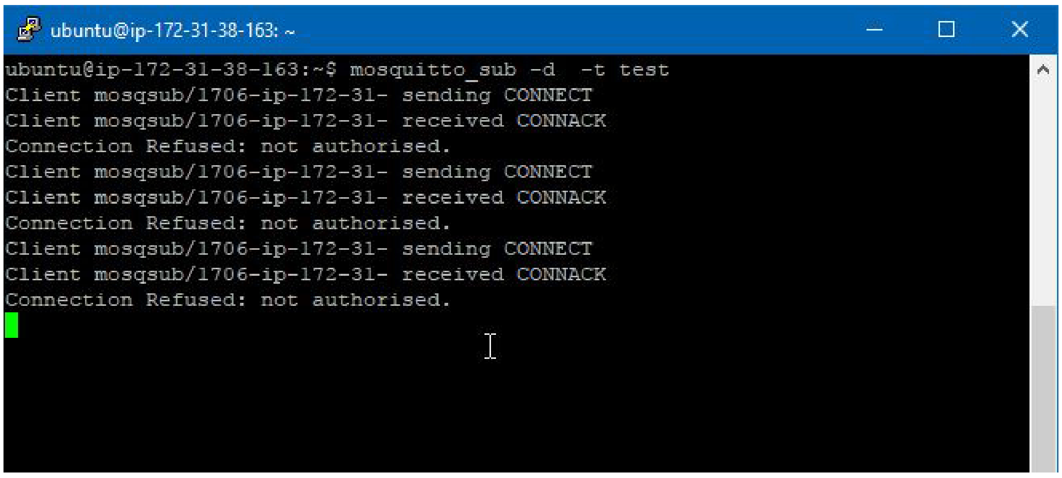
this line tells the server to install Mosquitto and mosquitto-clients, which enables the possibility to publish/subscribe through command line (see 3.16 and 3.17).
The default settings for the broker permits any user to publish and subscribe to a topic. To add a higher security level in the project is necessary to accept only certain connections. In our project we use a username and password, in order to accept only publications and subscriptions that have the correct parameters.
In figure 4.22 a publication without username and password is shown.



```
ubuntu@ip-172-31-38-163: ~
ubuntu@ip-172-31-38-163:~$ mosquitto_pub -d  -t test -m "Hello!"
Client mosqpub/1698-ip-172-31- sending CONNECT
Client mosqpub/1698-ip-172-31- received CONNACK
Connection Refused: not authorised.
Error: The connection was refused.
ubuntu@ip-172-31-38-163:~$ mosquitto_pub -d -u tesi -P 1802 -t test -m "Hello"
Client mosqpub/1699-ip-172-31- sending CONNECT
Client mosqpub/1699-ip-172-31- received CONNACK
Client mosqpub/1699-ip-172-31- sending PUBLISH (d0, q0, r0, m1, 'test', ... (5 b
ytes))
Client mosqpub/1699-ip-172-31- sending DISCONNECT
ubuntu@ip-172-31-38-163:~$
```

*Figure 4.22: MQTT publish without and with username and password.*

It can be seen that without the correct username and password the server doesn't accept the connection. In figure 4.23 is shown the subscription case.



```
ubuntu@ip-172-31-38-163: ~
ubuntu@ip-172-31-38-163:~$ mosquitto_sub -d  -t test
Client mosqsub/1706-ip-172-31- sending CONNECT
Client mosqsub/1706-ip-172-31- received CONNACK
Connection Refused: not authorised.
Client mosqsub/1706-ip-172-31- sending CONNECT
Client mosqsub/1706-ip-172-31- received CONNACK
Connection Refused: not authorised.
Client mosqsub/1706-ip-172-31- sending CONNECT
Client mosqsub/1706-ip-172-31- received CONNACK
Connection Refused: not authorised.
```

*Figure 4.23: MQTT subscription without the correct username and password.*

Brief explanation of the commands:

**mosquitto_pub** : it's the command to publish a message in a topic.

**mosquitto_sub** : command to subscribe to a topic

**-d** : indicate the debug-mode, the terminal will show the debug messages (in our case the CONNECT, CONNACK and PUBLISH messages).

**-t** : specifies the topic on which to publish the message, or to subscribe. This is wrote after the -t and, in our case, is "test".

**-m** : The message to be sent. In our case "Hello!"

**-u** : Specifies the username, that in our case is "tesi".

**-P** : The password's field, in our case "1802".

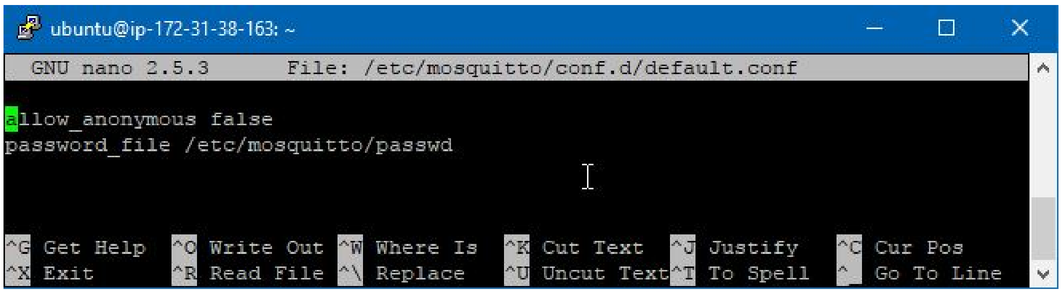
To configure Mosquitto to use passwords the first thing to do is create the password-file, and for this purpose Mosquitto includes a utility to generate it called "*mosquitto_passwd*". In our case:

*sudo mosquitto_passwd -c /etc/mosquitto/passwd tesi*

will create a file named "passwd" in the directory "/etc/mosquitto" with the password wrote after the command and the username "tesi".
Afterwards Mosquitto has to be configured in order to use the username and password. For this purpose the file default.config has to be modified like 4.24, where:

**allow_anonymous false** will disable all non-authenticated connections.

**password_file /etc/mosquitto/passwd** tells Mosquitto where to look for the user's password information.



```
  GNU nano 2.5.3      File: /etc/mosquitto/conf.d/default.conf

allow_anonymous false
password_file /etc/mosquitto/passwd



^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line
```

*Figure 4.24: Mosquitto's configuration file opened with nano editor.*

After this final procedure the MQTT broker is ready and reachable from the IP address 54.93.65.245, on the 1883 port.

### 4.2.3 Node-RED and its extensions

The last part of the server's set-up is the installation and configuration of the Node-RED instance. Thanks to the SSH and PuTTY (on Windows) the installation process is very simple and illustrated in the official Node-RED site[14]. It is similar to the installation of the MQTT broker, with two commands[15]:

*sudo apt-get install -y nodejs build-essential*

and

*sudo npm install -g node-red*

Once installed and started, the Node-RED editor can be found using a common web browser at: http://<server's-IP-address>:1880
Since the editor is on a public IP address, without an authentication method any user can, knowing the IP address of the server, access to the editor and modify the flows. To securing the editor's access it's possible to create a "login screen" (see 4.25) with a username and password. To enable this user authentication the "setting.js" file on

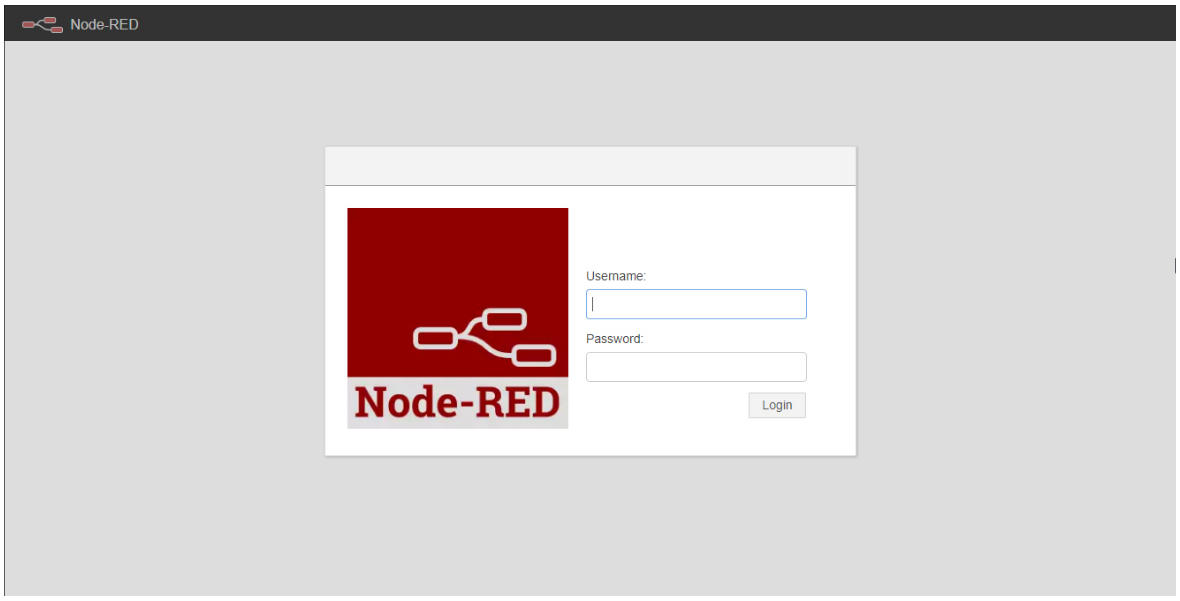
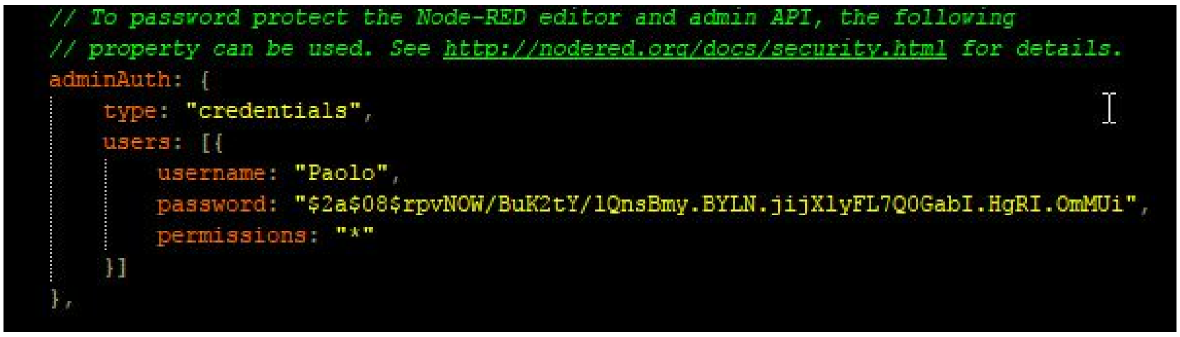


*Figure 4.25: Node-RED editor's login page.*

the Node-RED's directory is modified adding the lines in figure 4.26 where in the password's field is inserted the hash of the password obtain with the command:

*node-red-admin hash-pw*

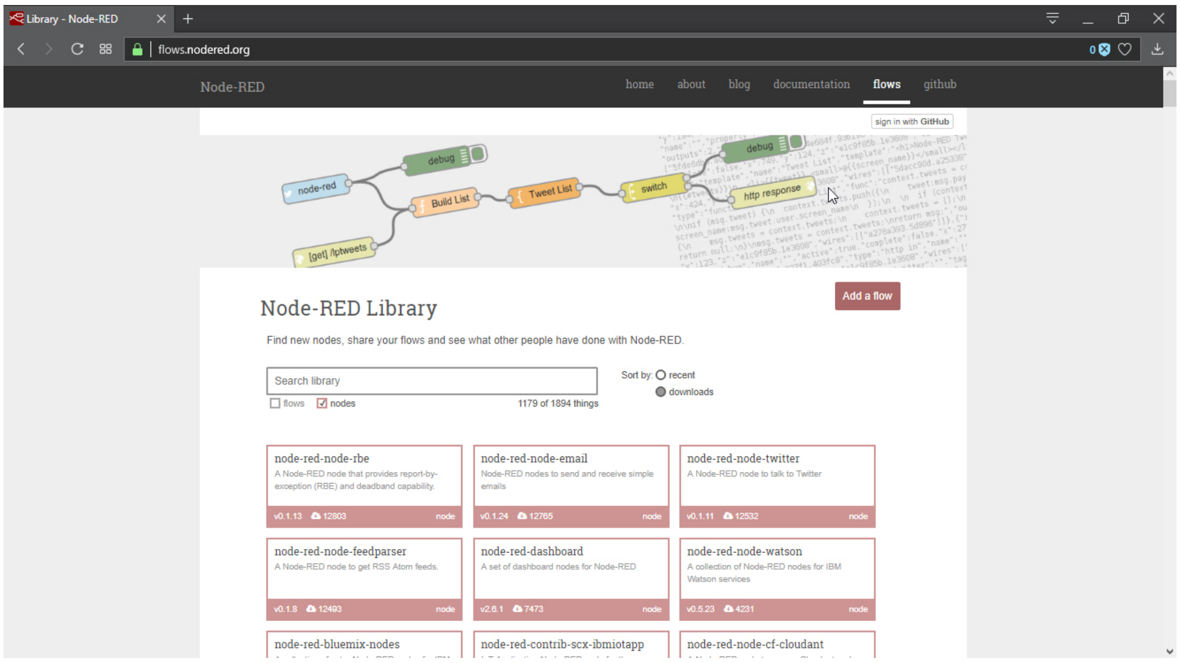
[14]https://nodered.org/docs/platforms/aws#running-on-aws-ec2-with-ubuntu
[15]For the commands related to the automatic start-up of the Node-RED instance see the website.



```
// To password protect the Node-RED editor and admin API, the following
// property can be used. See http://nodered.org/docs/security.html for details.
adminAuth: {
    type: "credentials",
    users: [{
        username: "Paolo",
        password: "$2a$08$rpvNOW/BuK2tY/1QnsBmy.BYLN.jijXlyFL7Q0GabI.HgRI.OmMUi",
        permissions: "*"
    }]
},
```

*Figure 4.26: Node-RED setting.js lines to add authentication.*

One of the advantages of the Node-RED platform is the vast quantity of nodes created by the community. There is a node for every kind of purposes (4.27), and it is simple to create one for our own purpose.



Node-RED Library

Find new nodes, share your flows and see what other people have done with Node-RED.

Search library

☐ flows ☑ nodes 1179 of 1894 things

Sort by: ○ recent ● downloads

node-red-node-rbe
A Node-RED node that provides report-by-exception (RBE) and deadband capability.
v0.1.13 12803 node

node-red-node-email
Node-RED nodes to send and receive simple emails
v0.1.24 12765 node

node-red-node-twitter
A Node-RED node to talk to Twitter
v0.1.11 12532 node

node-red-node-feedparser
A Node-RED node to get RSS Atom feeds.
v0.1.8 12493 node

node-red-dashboard
A set of dashboard nodes for Node-RED
v2.6.1 7473 node

node-red-node-watson
A collection of Node-RED nodes for IBM Watson services
v0.5.23 4231 node

node-red-bluemix-nodes

node-red-contrib-scx-ibmiotapp

node-red-node-cf-cloudant

*Figure 4.27: Node-RED vast library of nodes and flows from the community.*

In the project we used two principal set of nodes from the community: the Node-RED Dashboard and the Chatbot. The first one creates a set of nodes in the editor to quickly create a live data user interface with graphs, buttons, sliders and so on. The latter adds a set of nodes to visually build a chatbot[16] for Telegram, Facebook Messenger and Slack.
To install these two node sets the following code, written in the command line through SSH, is needed:

[16]A chatbot is a computer program designed to simulate a conversation with a human user. Chatbot are usually base on a mechanism of talk-reply, where the conversation is started by the user.

| | |
|---|---|
| *cd  /.node-red* | to go to Node-RED's installation folder |
| *npm install node-red-contrib-chatbot* | to install the chatbot's nodes |
| *npm i node-red-dashboard* | to instal the dashboard |

After the installation, the Dashboard's User Interface is reachable at the same address of the editor with "/ui" in the final part, e.g. in this project is *http://54.93.65.245:1880/ui*. In figure 4.28 the editor is shown, with this project's flows.

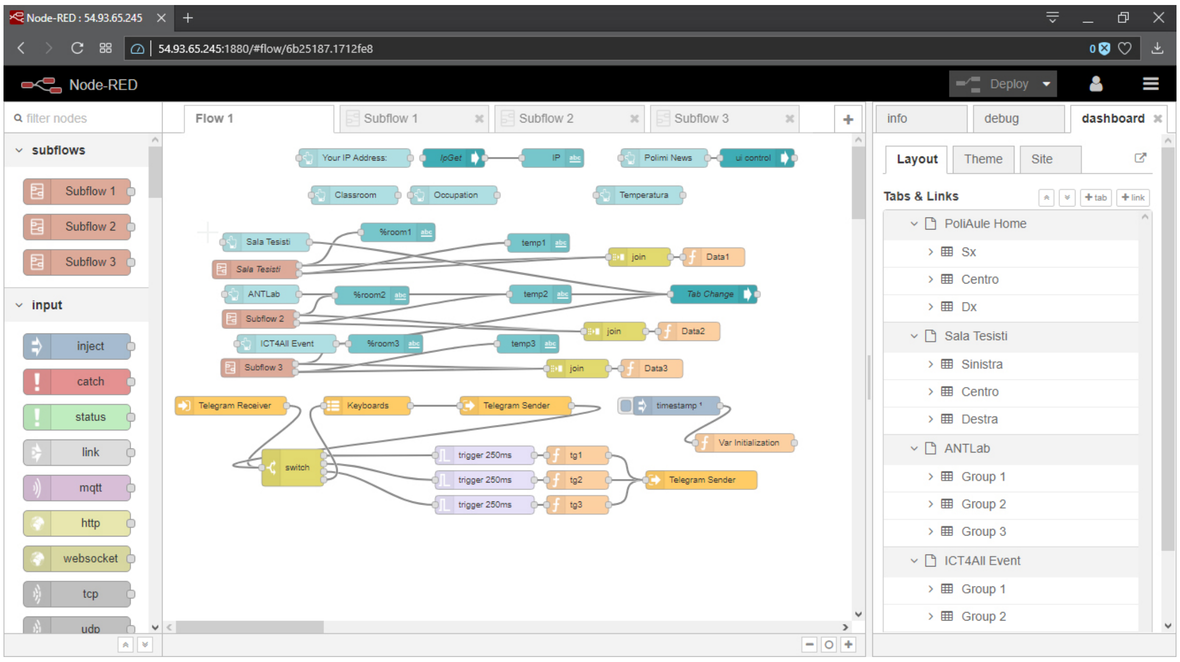


*Figure 4.28: Node-RED's editor page with the main project's flow.*

The installed nodes are on the left part of the editor, the flows are positioned in the centre and in the right bar an info,debug and dashboard tab are present. This last three tabs are useful to show the errors in the flows (the first two) and to show the position of the elements on the dashboard (the third one). Once the flow is ready and configured it can be "compiled" with the "deploy" button, the ui will be reset (if there are some data on the graphics) and the changes will take place.

### 4.2.4 The Final Flows

This final part of the chapter is the most important of the server-side implementation. It's the equivalent for the program inside the ESP, that is the brain that moves the hardware, but by the server side. This part is the Node-RED's flows implementation.
The project the Node-RED implementation is the key that puts a link between the hardware part and the visual-data part, where the user's can view the results of the estimation, the sensors' data and interact with the bot to retrieve information. The flow part is also the one that allow the administrator to interact with the ESP, giving the initials real number of persons in the room, and retrieving the estimations and values for the offline simulations. In the following part the structure of the flows are presented, with an explanation of their functionalities and results.
The structure of the program is divided in two four parts: one main flow, and three subflows representing the three rooms taking into account. Of course the number of subflows depends on the number of the rooms and it's easy to clone a subflow.
Let's start with an explanation of the subflows' structure. In figure 4.29 the first subflow can be seen (all the subflows have the same nodes and functionalities).

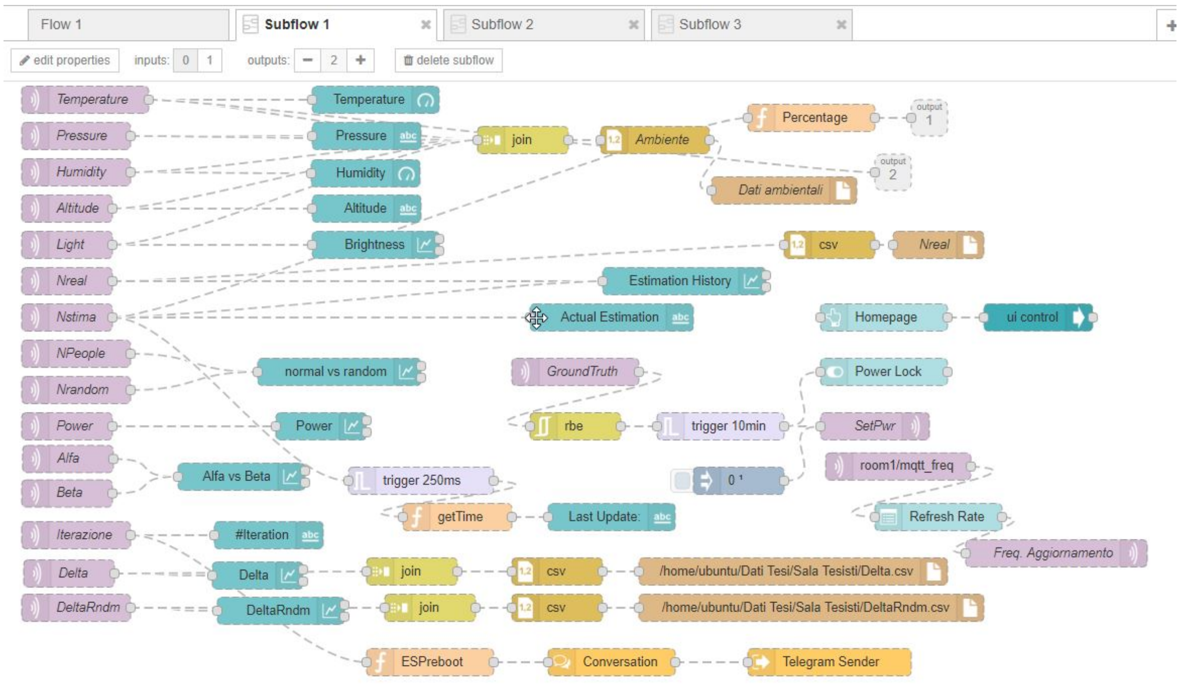
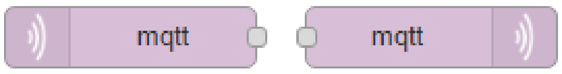


Figure 4.29: The first subflow represented by the tab "Sala Tesisti" on the Node-RED Dashboard

In the subflows there are various nodes of different colours. Every colour represent a type of node: liliac - the mqtt nodes, blue - dashboard graph's nodes, brown - the file

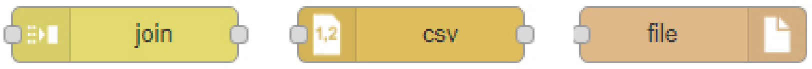
nodes, ochre - function's nodes, and so on.
As a matter of fact it is easier to explain the subflows dividing the nodes for their type:

### MQTT Nodes



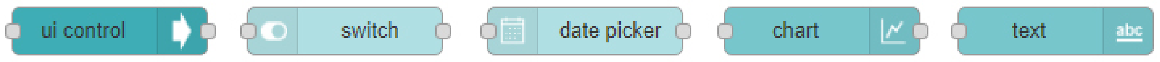
These nodes are the input (left) and the output (right) from an MQTT topic. They represent the link between the hardware part (ESP and sensors) and the user. In 4.29 these nodes are used to receive sensors' measurements, occupancy estimation and model's values. The MQTT outputs on the other hand, are used to set the PowerLock value (see flowchart 4.13), give the real number of people in the room (Ground Truth) and setting the send frequency value.

### Join-csv-file Nodes



These set of nodes are used, in 4.29, to generate a .csv file containing the sensor values, the Ground Truth values, and the number of captured probes divided by the power's range and their type of MACs (random or not).
The *join* node creates a raw containing the received values; the *csv* node creates a .csv file writing in append the values passed by the *join* node. At last the *file* node saves the .csv file in a chosen directory of the server, for offline analysis.

### Dashboard's nodes



The dashboard UI has dedicated nodes to display charts, button, sliders and so on in the UI page. In the project's flow the above nodes are used, where:

**ui control** receives as input the state of a button (Homepage, in 4.29) and change the tab if the button is pushed, bringing the user to the Homepage tab.

**switch** : prints out a switch on the page that change as the input changes. In our case it represents the PowerLock value.

**date picker** draws a drop-down menu where the user can select the value (in minutes) of the time between data-refresh. It has as input the value received by the MQTT topic, and as output the same MQTT topic, where the drop-down menu selection is sent. This "loop" is needed to keep memory of the last selected value when the dashboard restarts.

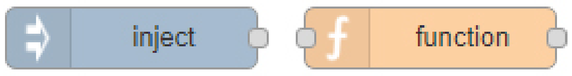
**chart** draws a graphic of the data in input. There are different types of charts, some of them accept more than one inputs and print it in different lines; other prints only one value at time. In 4.29 it's used several time to print out the values of the different data, from the temperature data, to a graph that shows the difference between estimated and real occupancy.

**text** this one is a box in which the numerical values are printed. In the flow is used to print out values such as the time of last update, the actual estimated occupancy, the pressure. the altitude, and the number of total sent data (#iteration).

### Trigger and Rbe nodes

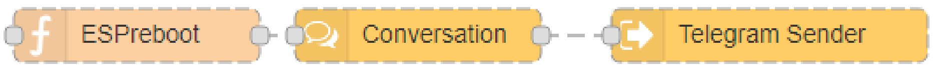
These nodes are used in cascade in our system. The *trigger* node sends a message on the output only if no data is received in the input after a certain amount of time. The Report By Exception (*Rbe*) node passes data from the input to the output only if its payload changes from the previous one.
In the project's flow these two nodes are used to automatically change the "PowerLock" value after 10 minutes from the last input. The *Rbe* receives as input the Ground Truth value (real number of persons) only if it's different from the previous one. Upon receiving the rbe output, the *trigger* sends a "0" value to its outputs node: the *mqtt-out* node and the dashboard's *switch* node. After 10 minutes of no input, the *trigger* sends to the "setpwr" topic and Power Lock switch the value "1". The *trigger* node is also used to update the time of the last update. When an input is received it activate a function that generates current date and time values to send to the *text* node of the dashboard.

### Inject and Function nodes



The *inject* node sends an output into the flow either manually or at regular intervals. In our case this node is used only at Node-RED start-up, and it sends a "0" value to the *mqtt* "SetPwr" node as a "reset" of the previous value.
The *function* node is "programmable" node: it's a JavaScript function block that runs the written function against the input value to generates an output. In our case this *function* node is used three times: the "Percentage" block receives as input the estimated number of people and sends as output the percentage of the room's occupancy based on the total number of seats in the room.
The "getTime" block is a block that sends as output a string with the current date and time only if an input is received.
At last, the ESPreboot function sends as output a message telling that the sensors has been rebooting, only if the MQTT input node "iteration" is equal to "1", meaning that the ESP is at his first transmission, so it's just powered up.

### Telegram set of nodes



This chain of nodes has the only function of telling if the ESP has rebooted by sending a Telegram[17] message only to the configured user. By default every message sent by the bot is received by any user who joint the bot's conversation. In this case the bot is configured to send this type of message, the reboot notice, only to one user identified by a "chatId" field in the message's packet.

All the three subflows are part of a single main flow, that is the main homepage of the site. Each subflow has two outputs (the grey boxes in the upper-right corner of figure 4.29): the first output gives the percentage of occupation of the subflow's room; the latter gives the temperature value, that is the main data people are interested about. The main flow is shown in figure 4.30. It can be seen that the output of the three subflows are connected to a dashboard's *text* node which shows their values. The *inject* node here is used to change tab (the Tab Change node) at a press of a *button* node; to get the IP address of the current visitor (IpGet node) and to redirect the user to the university homepage pressing on the button "Polimi News".
The major part of the main flow is the Telegram's bot implementation. The outputs of the subflows are joint and saved with a *join* and *function* node in global variables (data1, data2, data3 respectively). When a message is received the *Telegram Receiver*

[17]Telegram is a free messaging app available for all the major OS platform on the market.

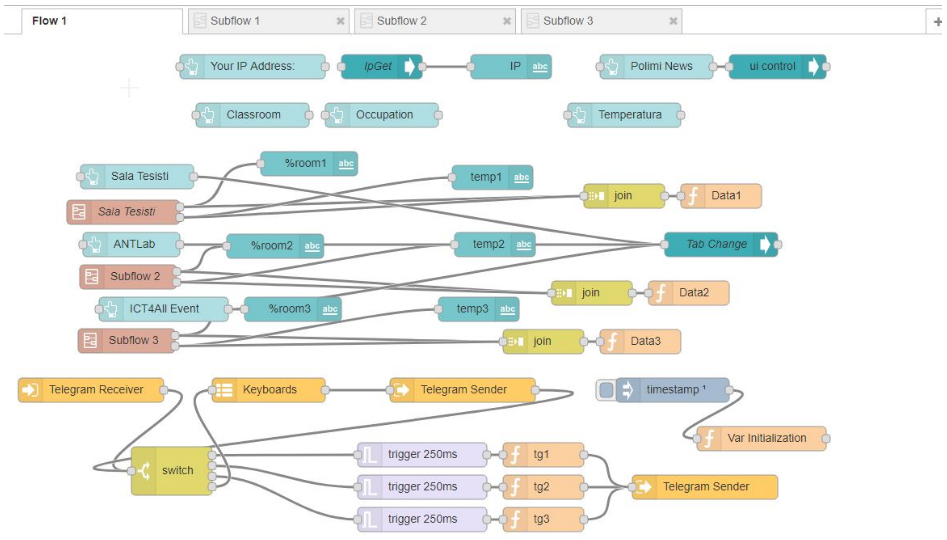
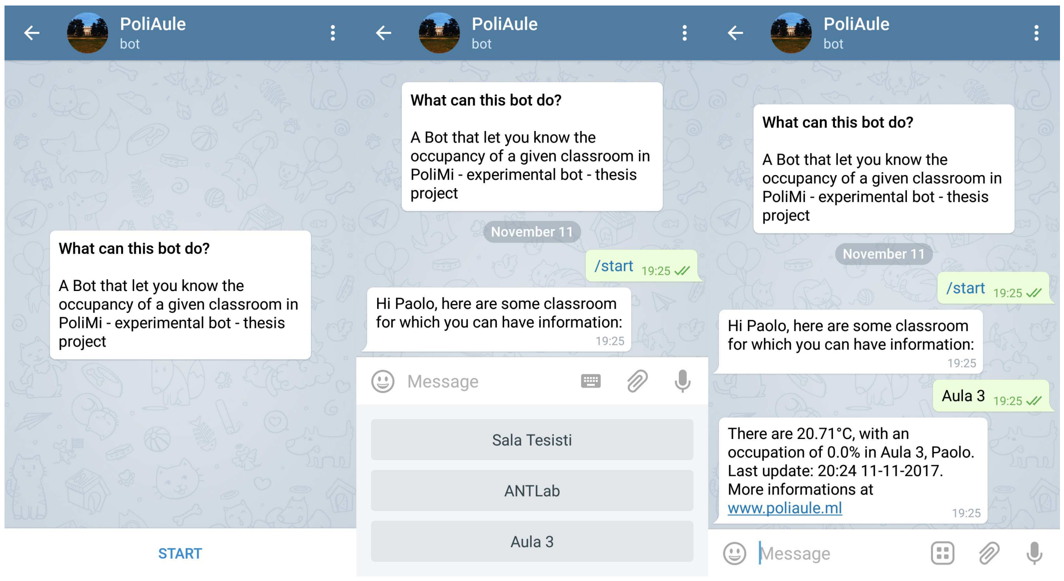


Figure 4.30: The Main flow of Node-RED

node send its content to a *switch* node. As the name suggest, the *switch* node selects the correct output based on the payload of the input message. As a default value it activates the *Keyboard* node which sends, through the *Telegram Sender* node, a keyboard layout in the user's chat, displaying the available class from which retrieve data. Since the project take into account three classes, there are another three outputs of the *switch* node, one per class. The functions next to the outputs prepare the packet payload format, including the data related to the choice, and sends them to the *Telegram Sender* node. The output of the conversation is shown in figure 4.31.



PoliAule
bot

What can this bot do?

A Bot that let you know the occupancy of a given classroom in PoliMi - experimental bot - thesis project

START

PoliAule
bot

What can this bot do?

A Bot that let you know the occupancy of a given classroom in PoliMi - experimental bot - thesis project

November 11

/start 19:25

Hi Paolo, here are some classroom for which you can have information: 19:25

Message

Sala Tesisti

ANTLab

Aula 3

PoliAule
bot

What can this bot do?

A Bot that let you know the occupancy of a given classroom in PoliMi - experimental bot - thesis project

November 11

/start 19:25

Hi Paolo, here are some classroom for which you can have information: 19:25

Aula 3 19:25

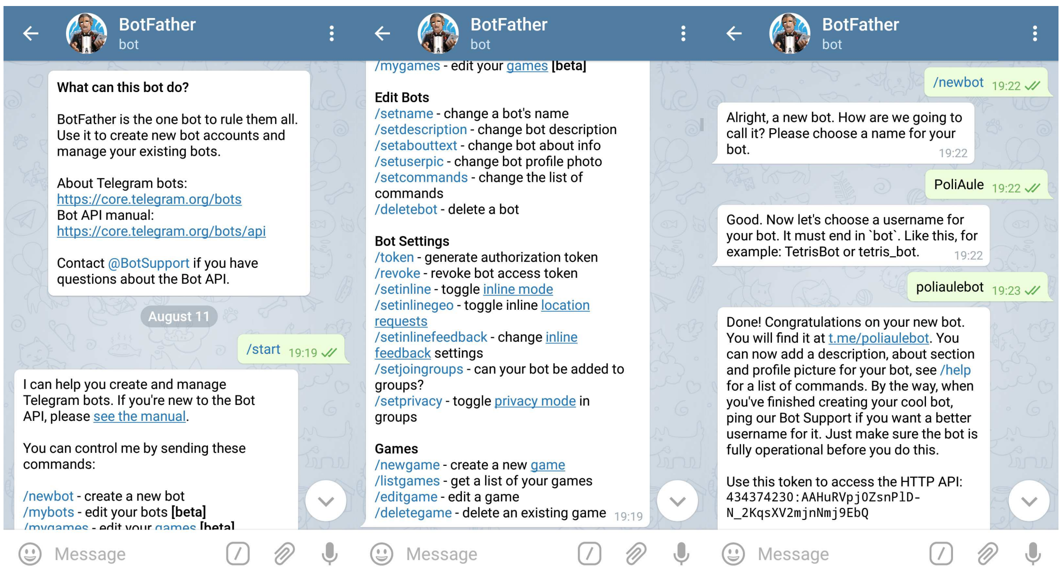
There are 20.71°C, with an occupation of 0.0% in Aula 3, Paolo. Last update: 20:24 11-11-2017. More informations at www.poliaule.ml 19:25

Message

Figure 4.31: A conversation with Telegram's bot implementation.

#### 4.2.4.1 Telegram Bot's creation and set-up

Telegram Bots are special accounts that do not require an additional phone number to set up. Messages, commands and requests sent by users to Bots passed through Telegram's servers that manages the encryption and the communication with the API. The server with the bot running communicate via HTTPS through, in our case, the Node-RED nodes.
All telegram nodes are configured to use an authorization token, which give access to the Bot's chat. To obtain the token, hence create a bot, a Telegram account is needed. Once having an account there is a Bot created with the purpose of create and manage the other bots called "BotFather". Following some simple steps, the bot will guide you in the creation of your own bot, setting the name and the picture and some other options. Some of the available commands on the BotFather are shown in picture 4.32, with some screenshots taken directly from the messaging app.


BotFather
bot

What can this bot do?

BotFather is the one bot to rule them all. Use it to create new bot accounts and manage your existing bots.

About Telegram bots:
https://core.telegram.org/bots
Bot API manual:
https://core.telegram.org/bots/api

Contact @BotSupport if you have questions about the Bot API.

August 11

/start 19:19

I can help you create and manage Telegram bots. If you're new to the Bot API, please see the manual.

You can control me by sending these commands:

/newbot - create a new bot
/mybots - edit your bots **[beta]**
/mygames - edit your games **[beta]**

Edit Bots
/setname - change a bot's name
/setdescription - change bot description
/setabouttext - change bot about info
/setuserpic - change bot profile photo
/setcommands - change the list of commands
/deletebot - delete a bot

Bot Settings
/token - generate authorization token
/revoke - revoke bot access token
/setinline - toggle inline mode
/setinlinegeo - toggle inline location requests
/setinlinefeedback - change inline feedback settings
/setjoingroups - can your bot be added to groups?
/setprivacy - toggle privacy mode in groups

Games
/newgame - create a new game
/listgames - get a list of your games
/editgame - edit a game
/deletegame - delete an existing game 19:19

/newbot 19:22

Alright, a new bot. How are we going to call it? Please choose a name for your bot. 19:22

PoliAule 19:22

Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris_bot. 19:22

poliaulebot 19:23

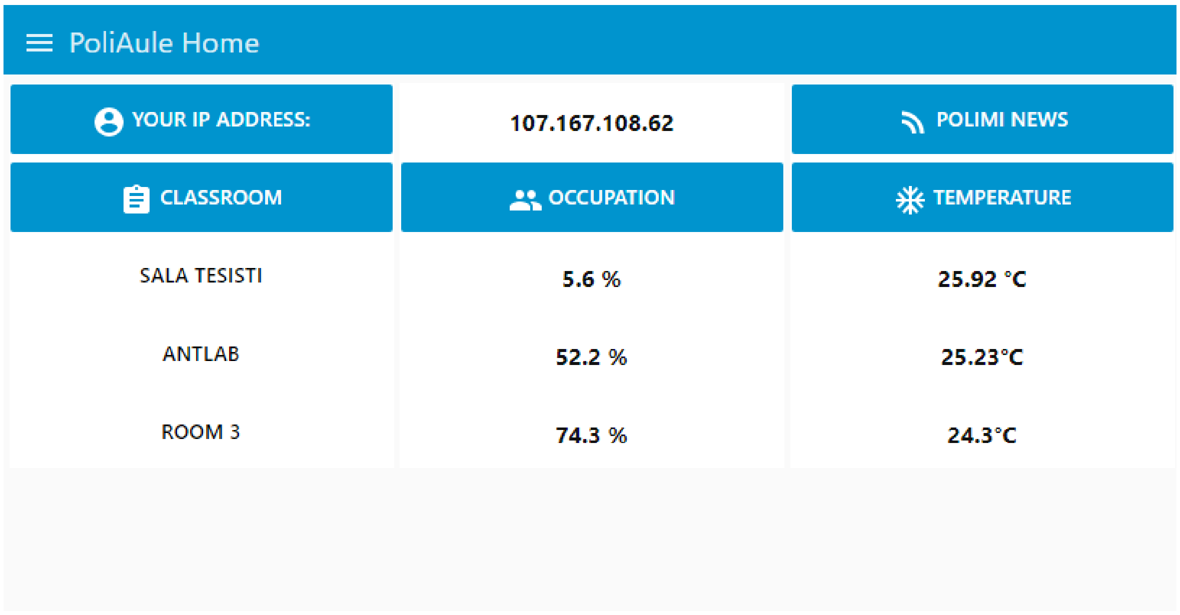
Done! Congratulations on your new bot. You will find it at t.me/poliaulebot. You can now add a description, about section and profile picture for your bot, see /help for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.

Use this token to access the HTTP API:
`434374230:AAHuRVpjOZsnPlD-N_2KqsXV2mjnNmj9EbQ`

*Figure 4.32: The BotFather's conversation guides through a bot creation.*

#### 4.2.4.2 Final Homepage

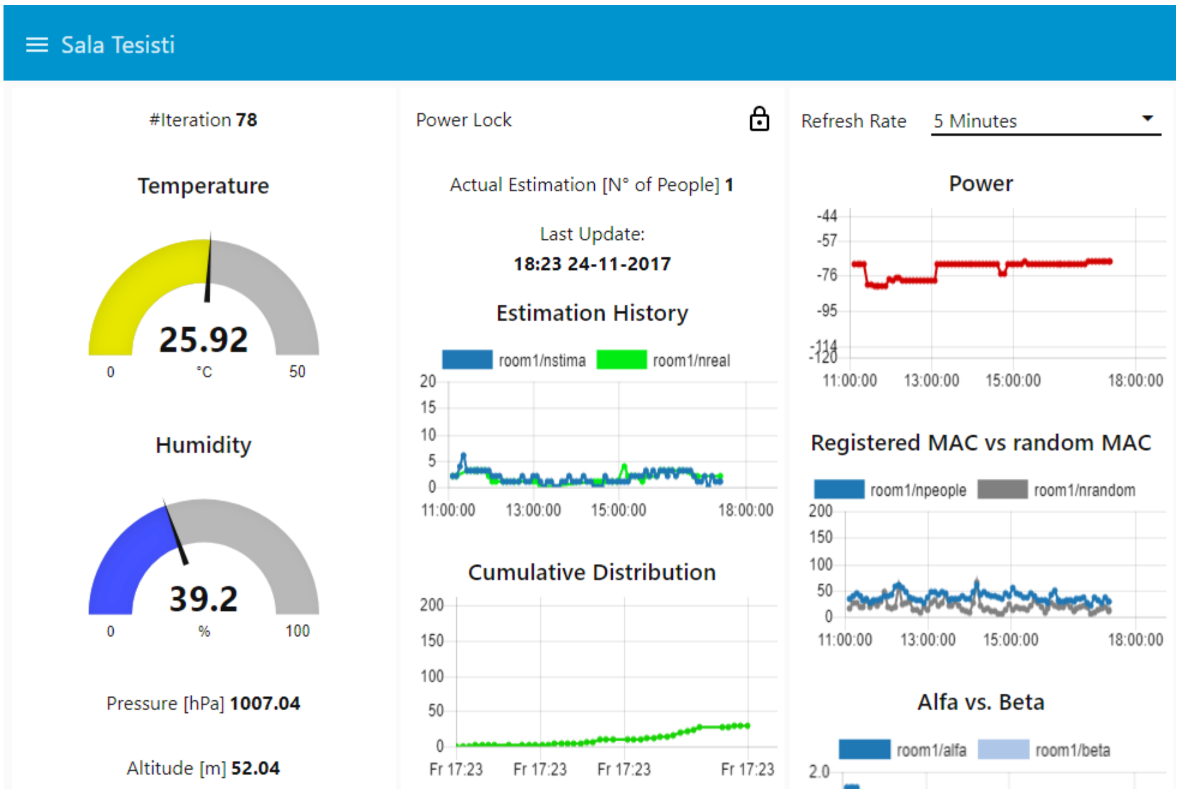
After the flows creation, thanks to a free domain subscription, the main site can be found at *www.poliaule.ml*. This visualization uses the Node-RED dashboard nodes, and enables the user to visualize the room's data and choose their update frequency.



PoliAule Home

| YOUR IP ADDRESS: | 107.167.108.62 | POLIMI NEWS |
|---|---|---|
| CLASSROOM | OCCUPATION | TEMPERATURE |
| SALA TESISTI | 5.6 % | 25.92 °C |
| ANTLAB | 52.2 % | 25.23°C |
| ROOM 3 | 74.3 % | 24.3°C |

*Figure 4.33: The main homepage*



Sala Tesisti

#Iteration **78**

Power Lock

Refresh Rate 5 Minutes

Actual Estimation [N° of People] **1**

Last Update:
**18:23 24-11-2017**

Pressure [hPa] **1007.04**

Altitude [m] **52.04**

*Figure 4.34: Data from the "Sala Tesisti" tab.*

# Chapter 5

# Experiments and results

This chapter is dedicated to the performance analysis of the previous discussed models (4.1.3.2). Thanks to the data provided by the sensor we are able to replicate the model's implementations in an "offline" mode, verifying its validity and trying to increase the performance in terms of estimation error. To replicate the models and analyse datas the Matlab (Matrix Laboratory) computing environment is used. Thanks to its powerful language and software, Matlab allow the creation of matrices, tables, graphs, simulations and so on. A sort of cross-validation approach (like in Machine Learning) is used to validate the models in different environments analysis.
The chapter is divided into three parts: in the first one the datasets used are presented, with a panoramic on the environments and methods used to get them. The second part is a panoramic on the model's validation technique used, with reference to the cross-validation. The third and most valuable part are the actual results of the model's validation, with the different characteristics and graphs explanations either for the final model and the "Model B" type of model.
Once the data are collected, through simulations the model can be optimized, in order to increase the performance of its error rate, and also the performance in terms of runtime for the code. For example, the code's runtime depends, as we've seen in 4.1.3.2, in large part by the model's function. If, through offline analysis, we reduce the range of the parameters $\alpha$ and $\beta$, the cycles will be shorter, and the program faster. Although this is an interesting solution, keep in mind that the optimization of the model based on a close dataset, can bring to the "overfitting problem". The model will adapt too close to a particular set of data, and will produce more errors (fails) in a different environment. The model will therefore "work well" only in certain particular conditions that is not the aim of this project. Trying to increase the model's performance through fitting the $\alpha$ and $\beta$ range on the detected room, increase our

results by a minimum of 0.01 to 0.1 units. Because of this results, and the overfitting problem, we decide to not include this part of the work in this thesis.

## 5.1 The Dataset

The data we're working on divides in two categories: the one taken from the developed sensor, and the one taken from a previous thesis work, with Wireshark.
The data taken from the sensor serves also as a "test" of the hardware and program's stability. It happen, in various occasion, that flaws on the code were found thanks to these tests. On this subject, one of the most critical problem was that, after some hours/days of working, the chip will casually restart. As explained in 4.1.3.2, after some debugging it becomes clear that this error depends on the WatchDog timer (3.4.2.1) that keeps expiring and resets automatically the board. This little problem, along with the change of the initial model and other minor bugs, is the main reason for the lack of a bigger dataset.
The data from the sensor were taken in two main places in Politecnico di Milano: the ANTLab, the Advanced Network Technologies LABoratory; an open-space dedicated to students that are about to graduate. The first area has about twenty always-on computers that uses the LAN network, and about 23 seats in a surface of 100m$^2$ (the "break room" is not counted). The latter has eighteen total seats, with a variable number of computer, usually connected over Wi-Fi. The observations in these sets are almost 200 (counted as the number of inputs "$N_T(t)$" given) either for the first and the second room.
As said above, we used also data from a previous research, from different rooms. Because of the lower number of inputs available, we decide to test only 2 of the rooms: the B.5.3 and L.26.01. The first one has a capacity of 80 seats, and the latter 70. Both, as opposed to the above rooms, are actual classrooms attended by students of different courses.

## 5.2 The Model's validation technique

The main purpose of the project was to estimate the number of persons in a room by their Wi-Fi devices. To achieve this goal we presented an estimation model 4.1.3.2 that "predicts" the occupancy of a room, and in order to test the accuracy of this predicting model, we use a sort of Cross-Validation technique. The dataset is therefore

divided in two part: the training set, that is the subset in which the training of the model is run. And the validation set, that is the remaining part of the total dataset with which the trained model is tested.
Since our hardware supports only a maximum "training" set of 40 data (due to hardware's limitation the ESP memorize only the last 40 inputs and apply the learning model only in that dataset), we implement a Matlab script that divides the total dataset in two part: the training phase is run over 40 inputs (in the case of a total input of 200) or 7 (in the case 10 total inputs), while the test phase is run over the remaining data. The validation process is run over either the final model and the model B of the estimation. To reduce variability multiple rounds of this cross-validation techniques are performed: in particular the program performs 500 rounds and, in every round, scrambles the order of the dataset in order to implement the training and test phase on different sets of data. The validation results are then combined (making the average) over the rounds to estimate the prediction model's performance.
This part basically replicates the chip's behaviour with different data inputs, in order to have a better view on the model's performance.

## 5.3 The Model's Performance

This section is divided into four part, one for each room analysed. Within each part, we show a confrontation between results of the final model, and the first model used (Model B, 4.1.3.2). Graphs and a final table (**??**) are included, in order to better explain and show the results.

### 5.3.1 ANTLab

The Advanced Network Technologies LABoratory is a research lab. of about 100m$^2$, with an adjacent smaller restroom. In our measurements we do not take in consideration the restroom part, but only the main lobby. We placed the sensor more or less in the middle of the room, and made 200 measurements along different days. Once the data are collected, we applied our model's validation technique described above and obtain the subsequent results.

**Estimation's Error**
This next graph 5.1 is divided in two part: the above one shows the confront between

the actual number of people and the estimated one. The bottom one shows the error, calculated as the absolute value of the difference between the actual and estimated number. The straight line in the second one is the mean value of this error. This chart

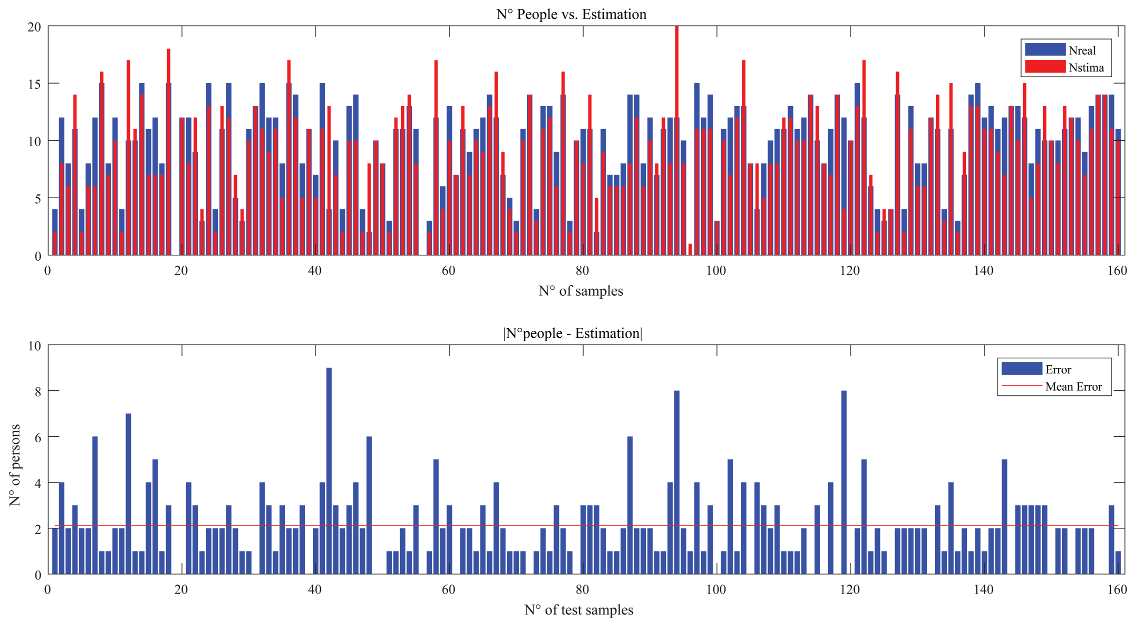


*Figure 5.1: Estimation vs. Real Value and Error graphs - Related to the ANTLab*

represents one single round of the 500 performed by the validation program. In this case the maximum error was of 8 people, the minimum of 0 and the MAE - Mean Absolute Error (the red line of the second graph) was 2.1187 (2 if we approximate to the nearest integer). Considering that the total number of seats in the Lab. is 21, 2 persons out of 21 seats makes an average error rate of 10%.

### Model's parameters

In graph 5.2 the model's parameters are shown: these are the variables calculated in the training part of the model, among the 40 inputs. At the beginning the algorithm tries to find the values of $\alpha$, $\beta$ and Power that minimizes the RMSE (the upper left graph). After 40 inputs the last parameters found are supposed to be the best ones, and used in the test phase among the remaining part of the dataset (160 elements in our case). In this particular round, the optimal values are: $\alpha$=0.6, $\beta$=0 and Power=-79dB. In this case the final model and Model B are equal, since the $\beta$ used in the test part is equal to zero. From the charts it can be seen that the first values of these three parameters have a higher variance with respect to the final values. This is a consequence of the estimation model's function, that reduce the variance taking into

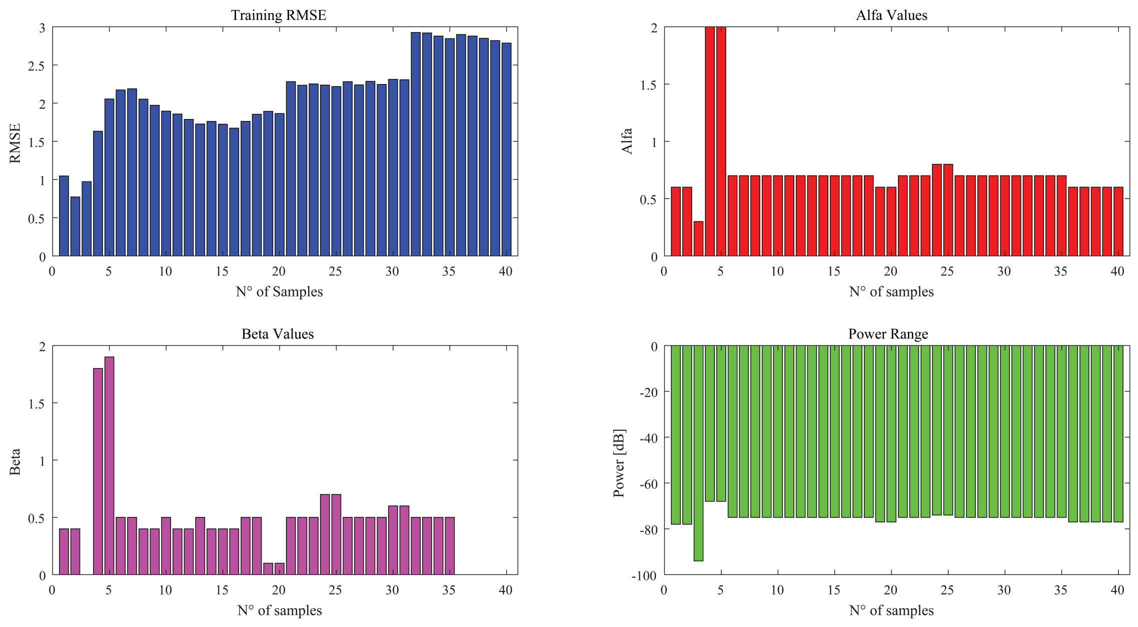


*Figure 5.2: Model's parameters in one round.*

account the past values of the parameters and inputs. As explained in 4.1.3.2, every new choice of parameters has to consider also the past inputs, trying to be suitable for the present and past measurements. The consequence is that, after some initial changes, the parameters begin to stabilized to a fixed or few amount of near values. In this case, without considering the first 10 values, $\alpha$ is in a range between 0.6 and 0.8, the power between -73 and -77dB, and $\beta$ between 0 and 0.7.

The $\beta$ parameter is the one with more variability because of its nature: it's the coefficient of the random MACs count that has, itself, a high variability. As a matter of fact, the graph 5.3 shows the evolution of the total number of random and non-random MAC addresses seen by the ESP during time. It can be seen that the variance of the Random MACs count is higher than the non-random ones. In particular, the first is around 146 and the latter around 75.

**Final Comparison**

The shown values are taken from a single round of the validation algorithm. As explain in the previous section, our cross-validation technique runs 500 rounds, each one with different training values.

In graph 5.4 a comparison between the final and the B model is shown, along the 500 rounds.

Although is hardly visible, the MAE value of the final model is higher than the one from the model B. The first one is 2.2257 while the second one 2.1963. As expected

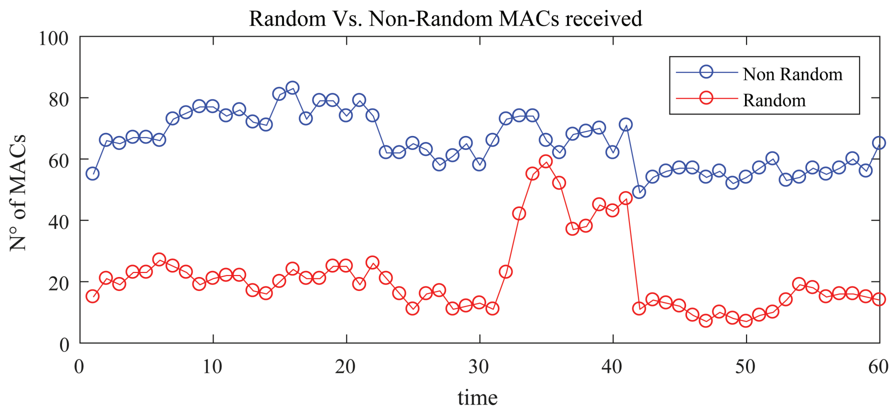


*Figure 5.3: Evolution of the number of MACs, divided by type. A range of 50 inputs is considered.*

even the RMSE is higher in the final model, with values of 2.21 and 2.18 respectively. Of this results we calculate also the maximum error committed. In the final model the maximum error reaches 15 persons, against the model B's 14. Of course this maximum value cannot be seen in the graph 5.4, because it only represents the MAE value of each round, so it's calculated memorizing the maximum value obtain in each round and then taking the maximum of all 500 rounds.

Having a more precise average error, the percentages of the two models, with respect to the whole capacity, are 10.6% and 10.46% respectively.

In the next graphs, 5.5 and 5.6, the parameters values can be seen. Also in this case, there are slightly different values between the two models.

The first set of graphs shows the confrontation between $\alpha$ and Power values of the two models. Though again is difficult to see, the mean values are slightly different. From the final model we have $\alpha$=0.545 and Power=-79.4480, while from the model B we have $\alpha$=0.528 and Power=-79.8140. Of course these different values are due to different $\beta$ values, as seen in the 5.6 graph. Although the MAE and RMSE are slightly better in the case of the first model, the final one remain the best choice relating to the possible future scenario.

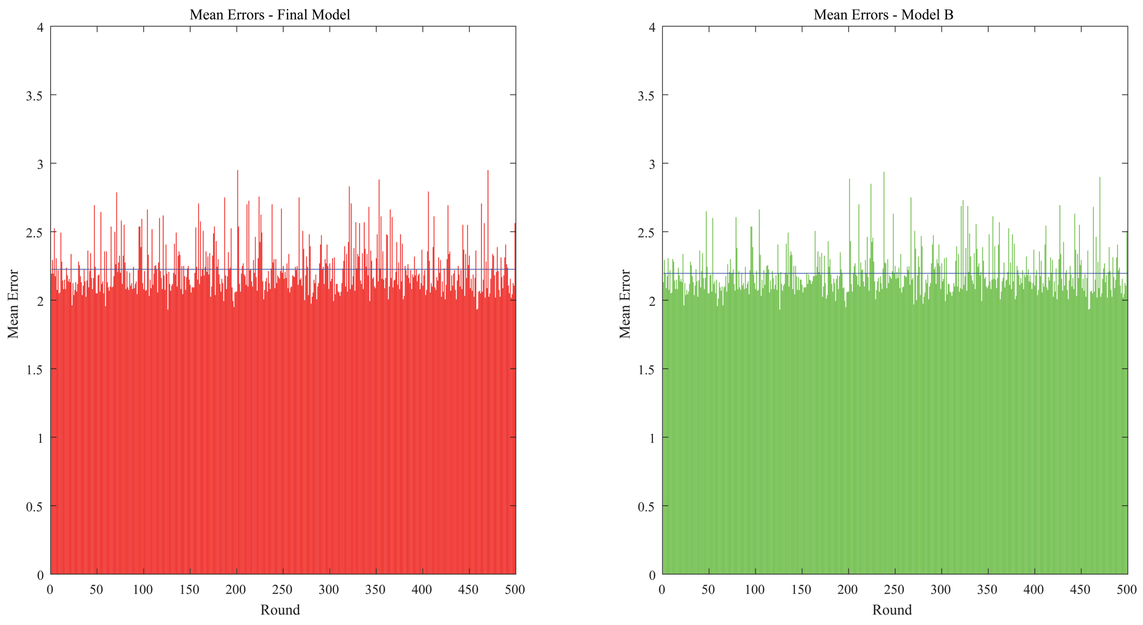


*Figure 5.4: Error graphs of the first and final model.*

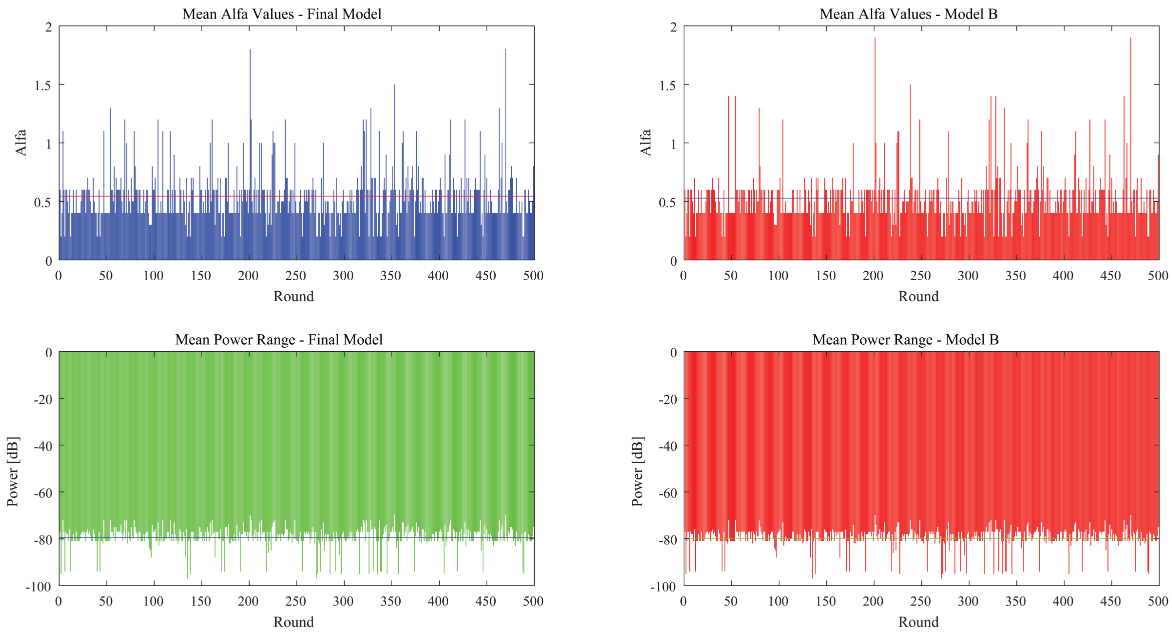


*Figure 5.5: Model's parameters confrontation.*

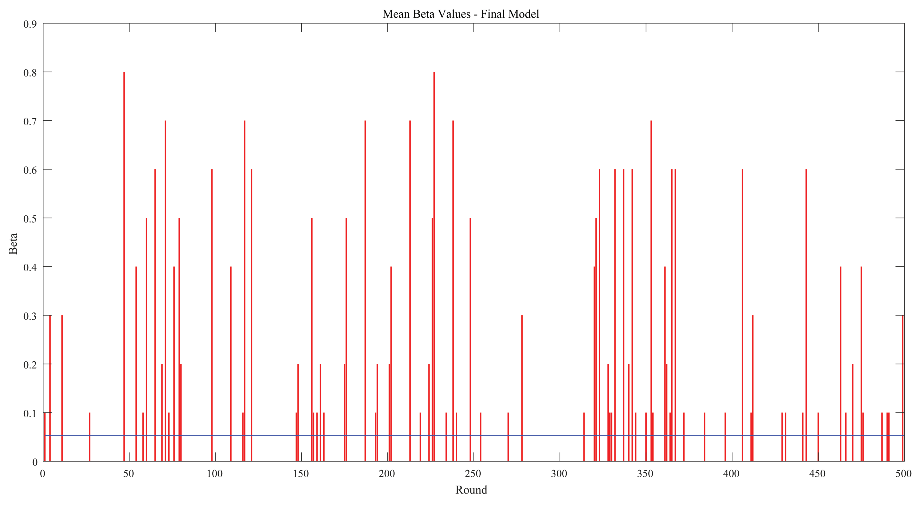


*Figure 5.6: The $\beta$ parameter variations with rounds.*

### 5.3.2 Sala Tesisti - OpenSpace room

This room is an open space of more or less 21m$^2$, and 16 seats. As in ANTLab, we tried to place the sensor in the middle of the room to make measurements. Although smaller, this room is characterized with a high rate of people that comes and goes, but a lower mean number of total presence.

#### Estimation's Error

Like in the previous subsection, the first graph presented is the one related to the error rate. The first one represent actual amount of people (blue line) in contrast with the estimated number (red line). The second graph represent the error of this particular round (that can be lower or higher in different rounds). This time the error values are a

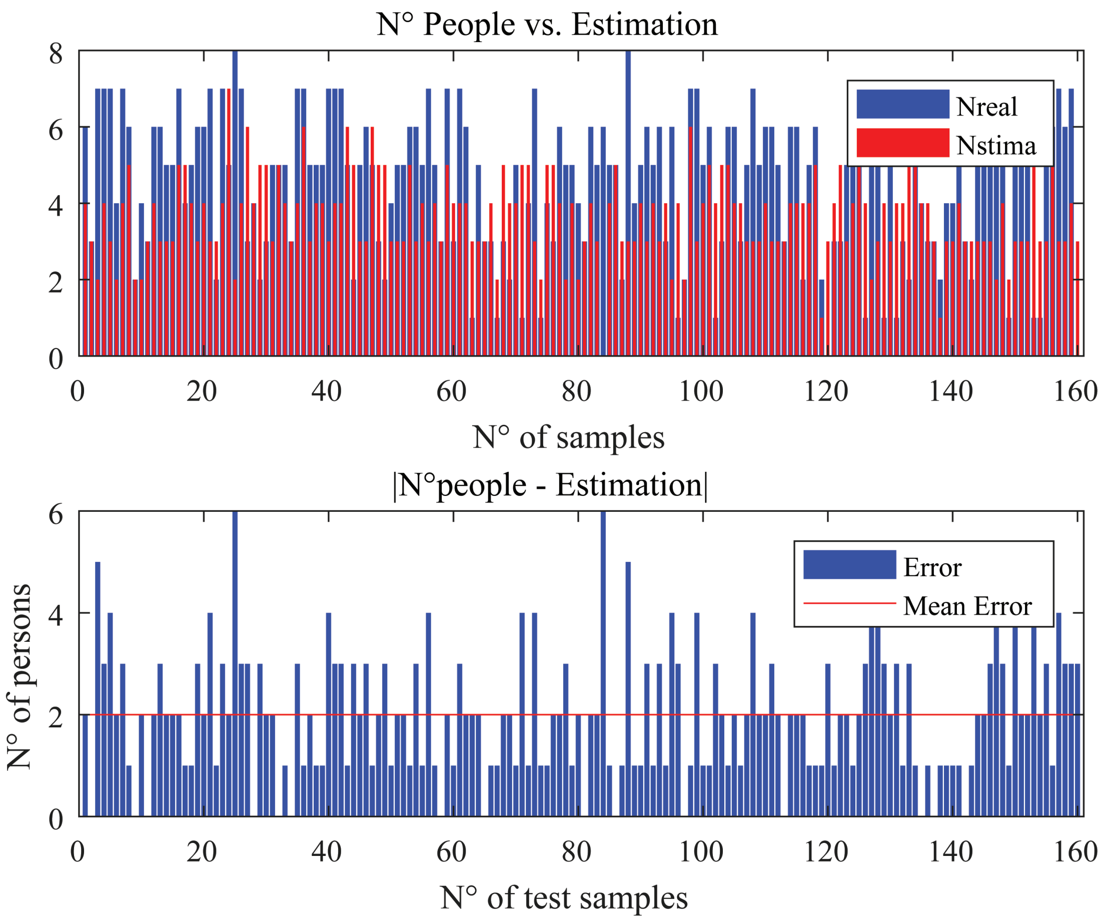


*Figure 5.7: Estimation vs. Real Value and Error graphs - Related to the open space.*

little different. The maximum error of this round is 6, while the minimum is obviously 0. The mean error is 2 that, with respect to the capacity, correspond to an error of

11%. In this single round the RMSE is 2.07, similar to the MAE value.

## Model's parameters

In the 5.8 chart the parameters of the training phase are shown. Thanks to the model,

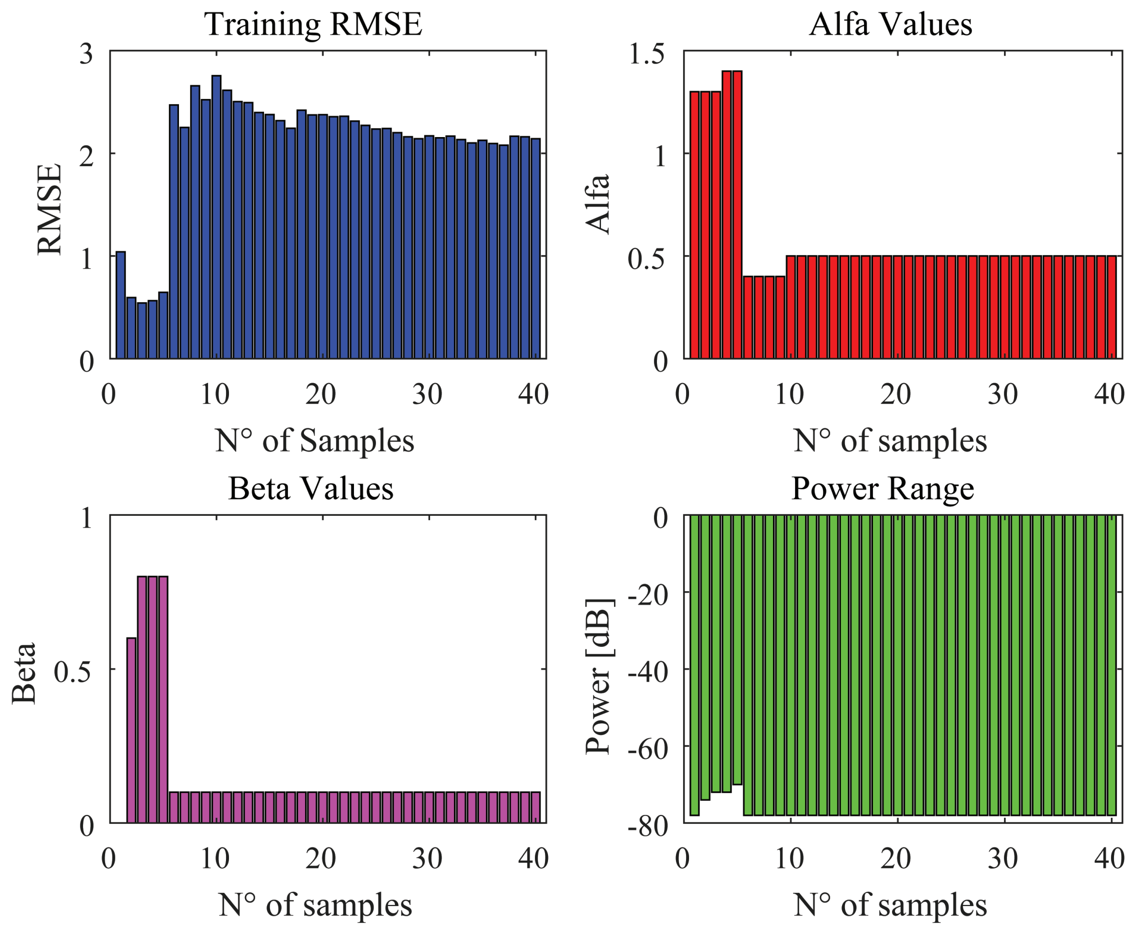


*Figure 5.8: Model's parameters of the previous round.*

we can see the same behaviour of the previous charts: at the beginning the parameters' values have a high variance, but after some inputs they begin to settle down. In this round, $\alpha$ and $\beta$ at the end of the training part are 0.5 and 0.1 respectively. The power settles to -78dB.

## Final Comparison

Running 500 rounds with this dataset gives better results with respect to a single round. If in the previous charts the error of the final model was 2, after 500 rounds the MAE results 1.72 and 1.70 respectively to the final and model B variant. Even in this case the model B performs slightly better than the final model. In terms of RMSE the

two models reach 1.61 and 1.57.
In 5.9 the comparison between the two errors can be seen. The maximum error is of 9

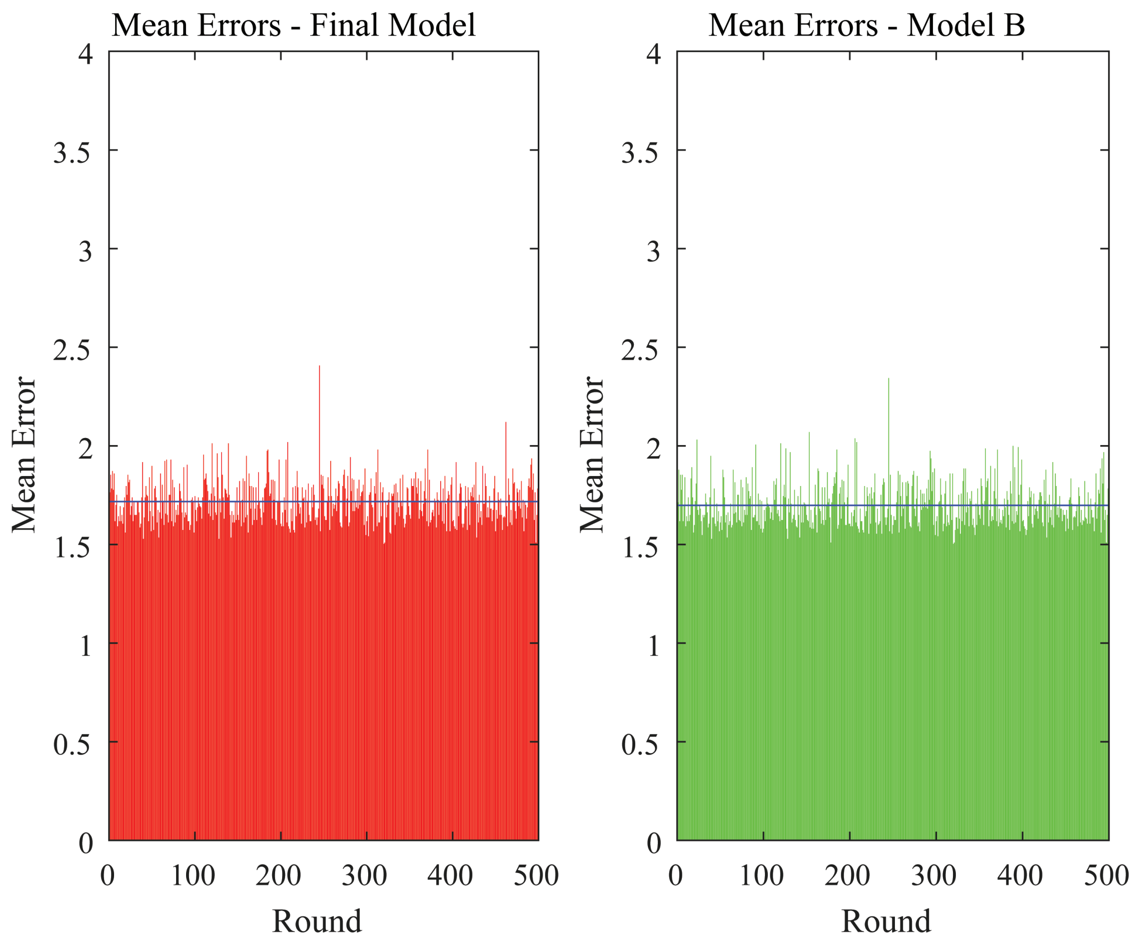


*Figure 5.9: Error graphs of the first and final model -Open space.*

and 8 persons for respectively the final and B model. The percentage error with respect to the capacity of the room is respectively 10.75% and 10.62%, calculated using the MAE and not the RMSE. It can be seen that the difference between the two models is negligible.
In charts 5.10 and 5.11, the difference of parameters values can be seen. It can be seen that, since the power range of the first model is wider (-82.62dB on average), its $\alpha$ values are lower with respect to the final model. In particular we have an average $\alpha$ value of 0.47 and 0.71 respectively. In the final model the average power is -79.14dB, lower than the one in model B but balanced with the higher value of $\alpha$ and $\beta$ (this last one is 0.15 on average).

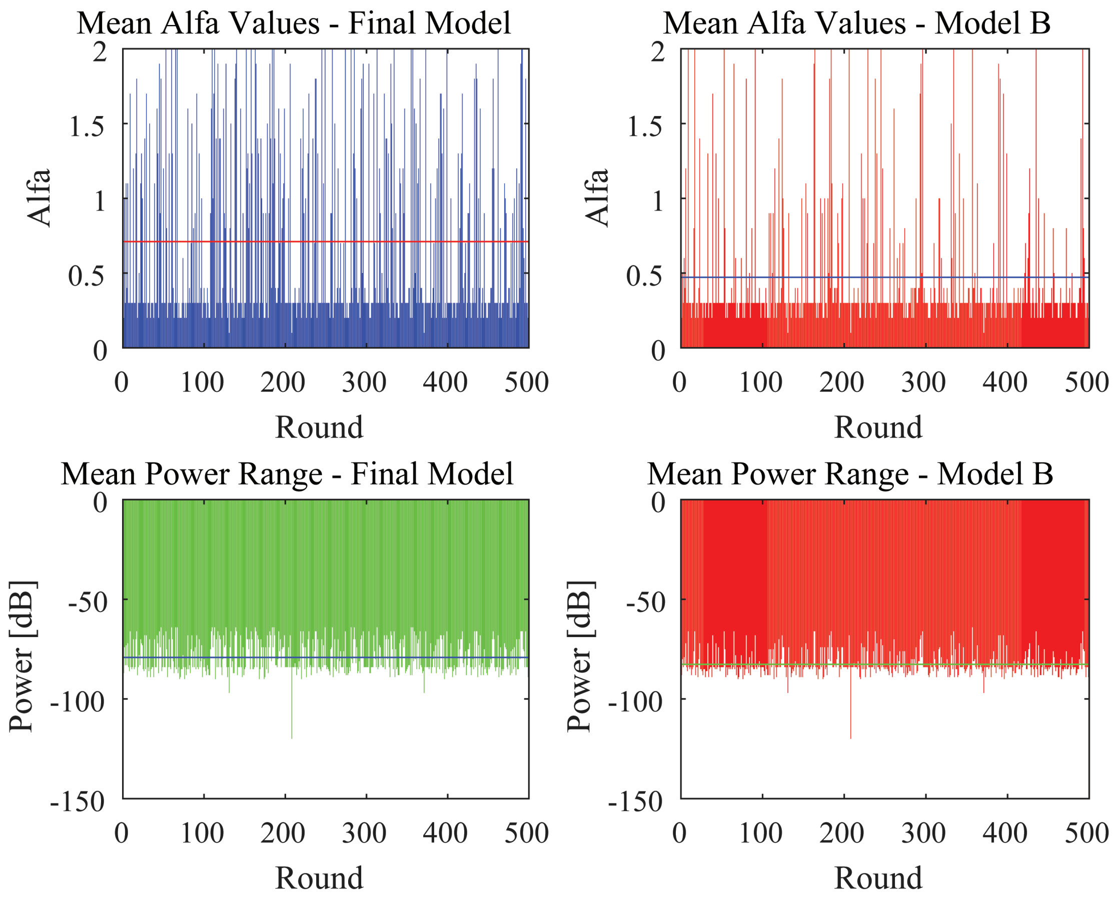


*Figure 5.10: Model's parameters confrontation - Open space case.*

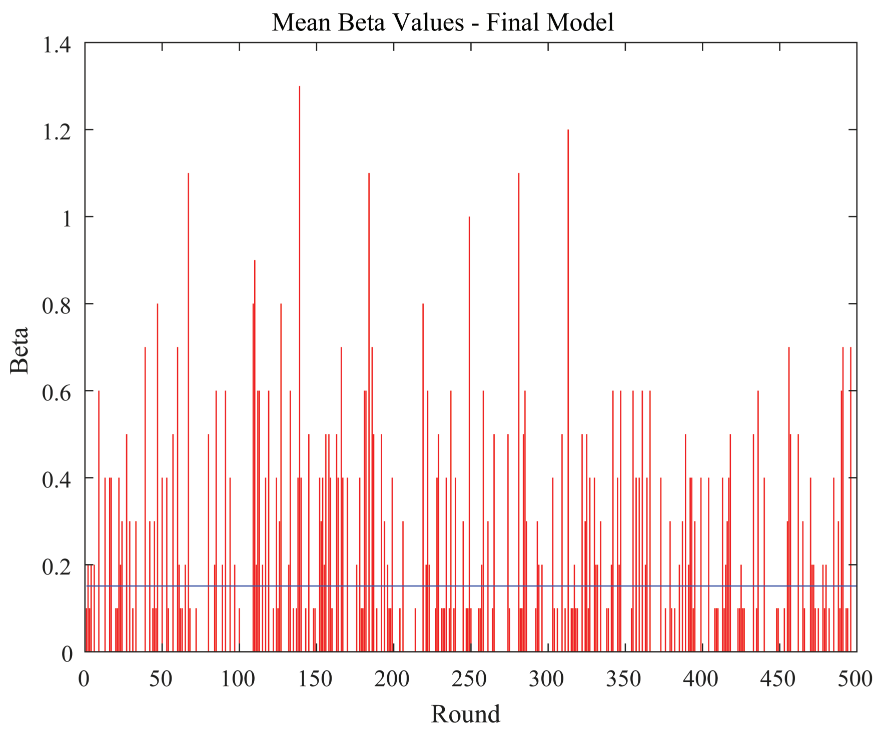


*Figure 5.11: The $\beta$ parameter variations with rounds, and its average - Open space case.*

### 5.3.3 B.5.3

This is a classroom with a capacity of 80 seats. Unlike the above rooms, this one is placed in a very crowded building during lesson periods. It's on the fifth floor of a building, so it receives "noise" (unwanted probes) also from the others floors (the rooms before received noise only from the upper or lower floor, but not from both). From this classroom we are able to capture only 11 inputs. Of this 11 inputs, 7 are used for the training part, while the other 4 for testing.
Since the dataset is smaller, we propose directly the output of the validation model after 500 rounds, with the comparison with the two models. In the first chart 5.12 a comparison between the error of the two models is shown.

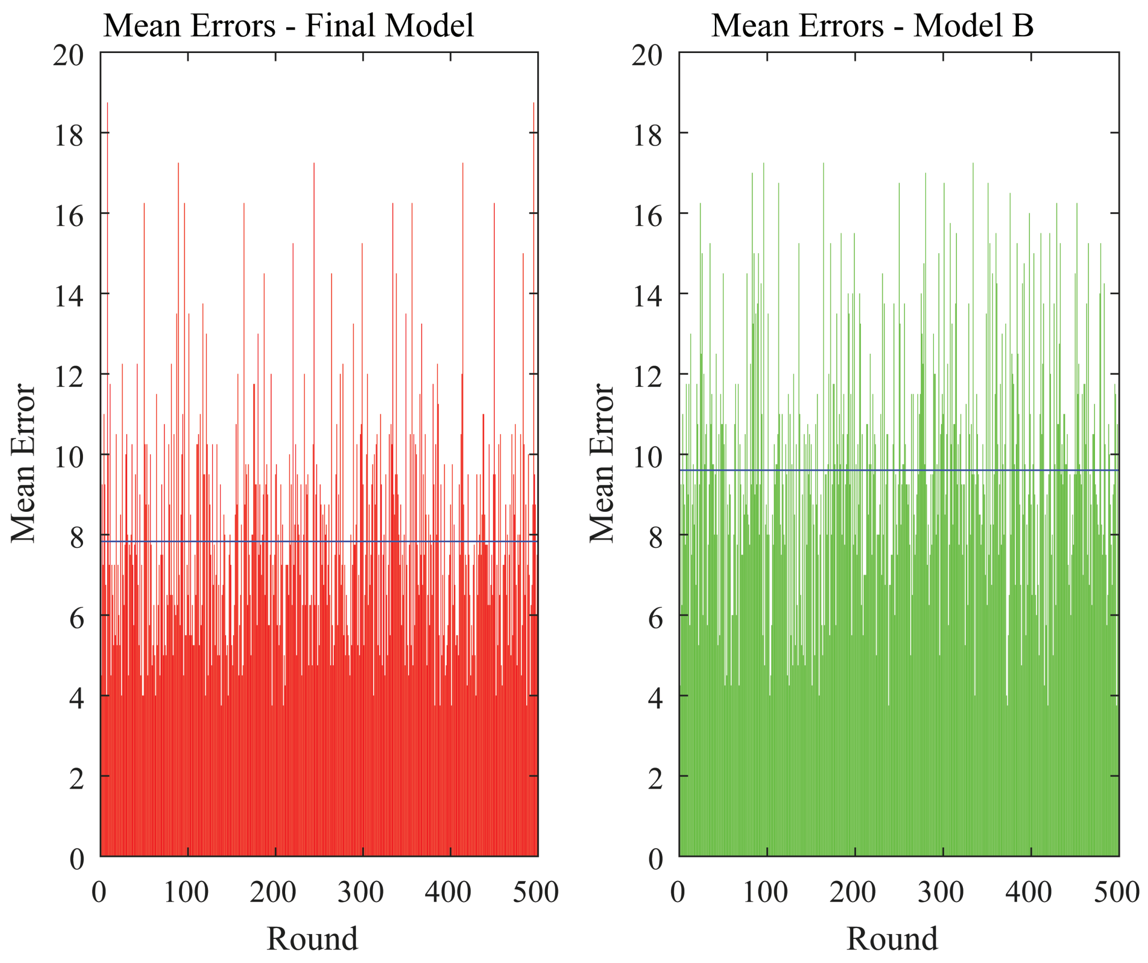


*Figure 5.12: Error graphs of the first and final model - Room B.5.3.*

In this case the average error of the final model is clearly smaller than the one from the first model. While the first one is 7.8, the latter stops at 9.7. The RMSE of the two models are 4.14 and 5.3 respectively for the final and B model. Compared to the whole capacity of the classroom, these MAE translated to a percentage error of 9.75% and 12%.

In the next graphs we can see the differences in the parameters' choices. In 5.13 we can

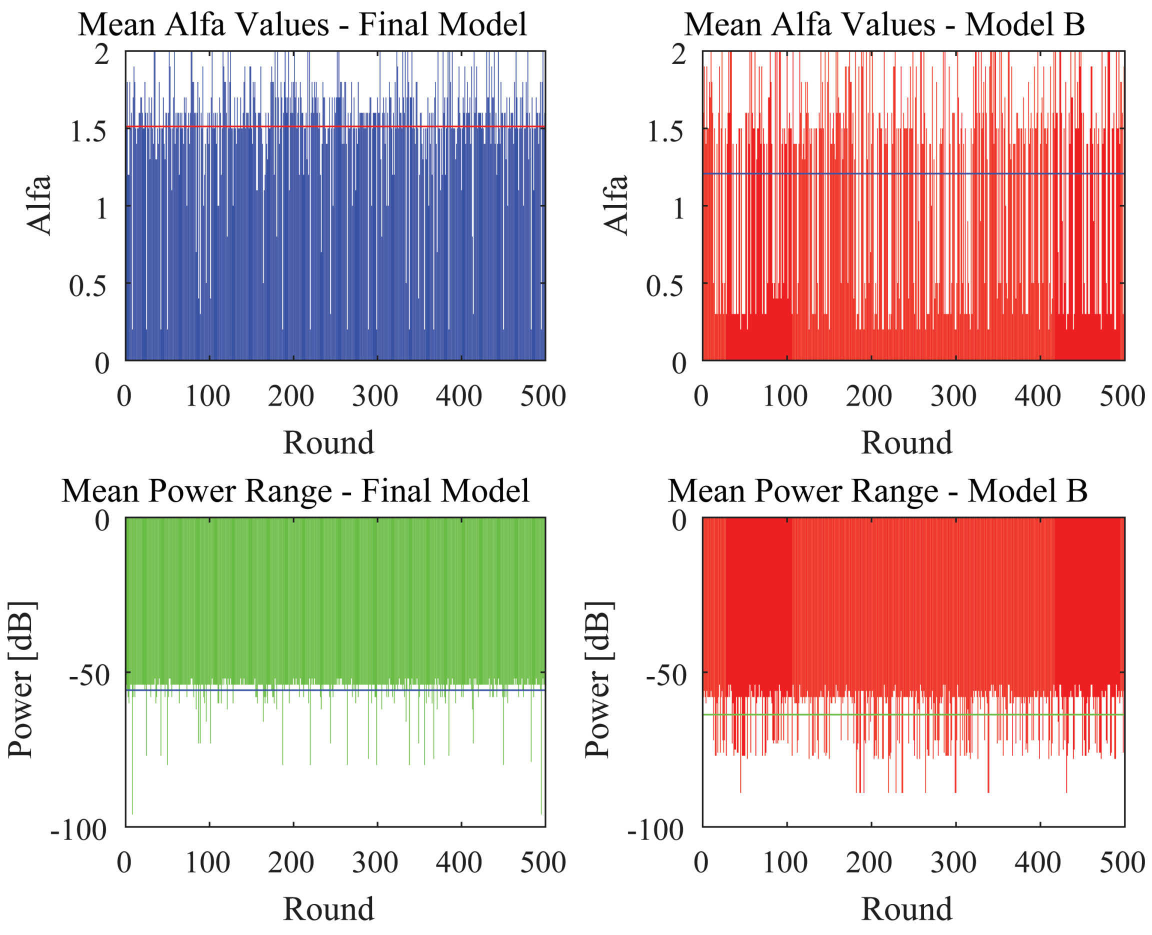


*Figure 5.13: Model's parameters confrontation - B.5.3 Classroom.*

see that the $\alpha$ value of the final model is higher with respect to the B model, but the power range used is lower. For the final model the average power used is -55dB, with coefficients $\alpha$ and $\beta$ equal, on average, to 1.5 and 1 respectively. The $\alpha$ on the other model is 1.2, with an average power value of -63.7dB. Unlike before, with this dataset the value of $\beta$ increases till 1.03. This means that in this case the count of random MACs gives a high contribute to the final estimation.

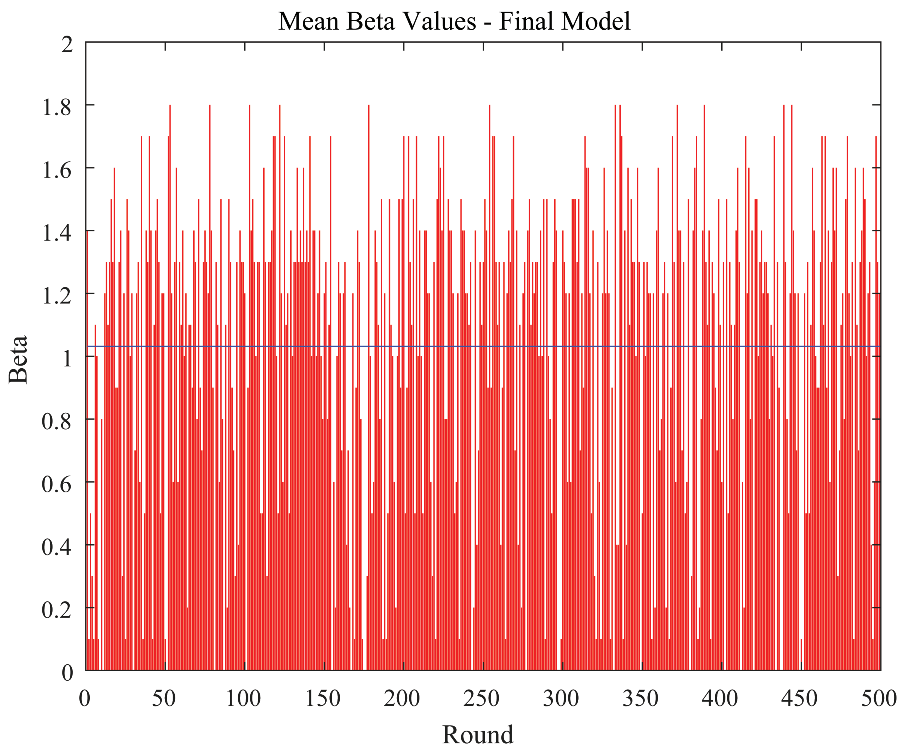


*Figure 5.14: The $\beta$ parameter variations with rounds, and its average - B.5.3 Classroom.*

### 5.3.4 L.26.01

This last room is a classroom located near the dining hall of the university. It's on the basement of the building, and can be affected by the noise coming from the more crowded upper floor. The classroom has a capacity of 70 seats.
For this classroom our dataset is composed of 10 measurements: 7 inputs used for training the model, and the remaining 3 to test it. Giving a look to the results of 5.15 it can be seen that the first model gives a slightly better performance with respect to the final one. When the first one reach an average error of 4 persons, the final one performs poorly, reaching a value of 4.37. Translated in percentage with respect to the capacity of the classroom, the errors are of the 5.7 and 6.24% respectively. The RMSE is placed at 1.92 for the B model, and 2.04 for the final model.
In 5.16 and 5.17 there are the usual graphs shown the parameters of the models. In this case the values of $\alpha$ are nearly the same, with values of 1.294 and 1.264 respectively in the final and B model. Even the mean powers are really similar, with -59.18dB and -59.23dB values. In conclusion, the different values of error are produced by different values of $\beta$ during the various rounds. Even if the average value is 0.08, $\beta$ assumes values different from zero in various rounds.

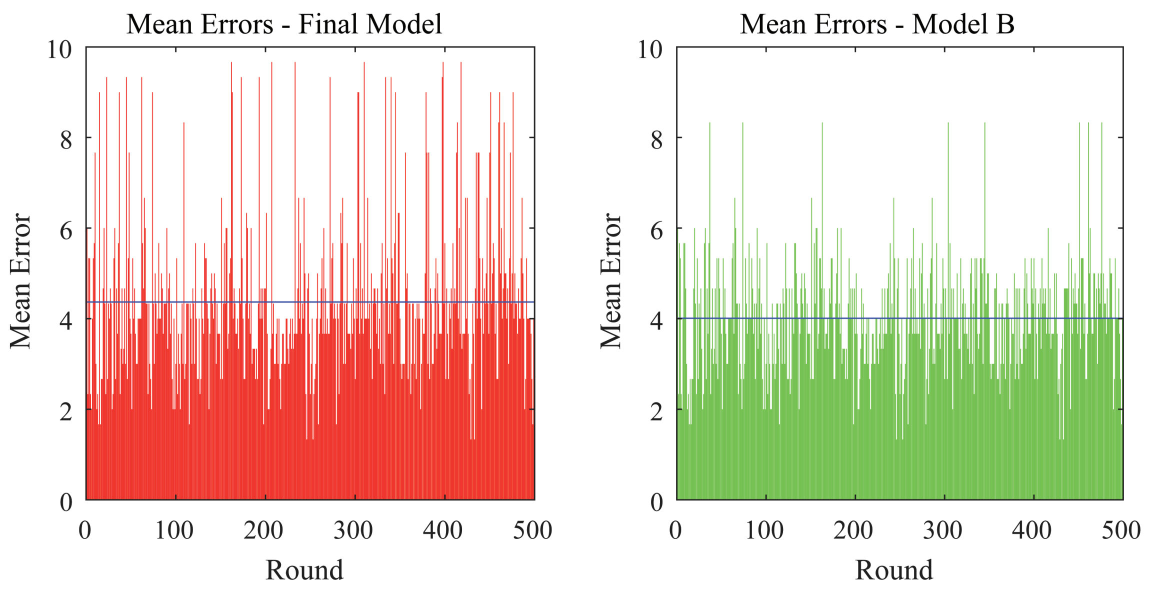


*Figure 5.15: Error graphs of the first and final model - L.26.01*

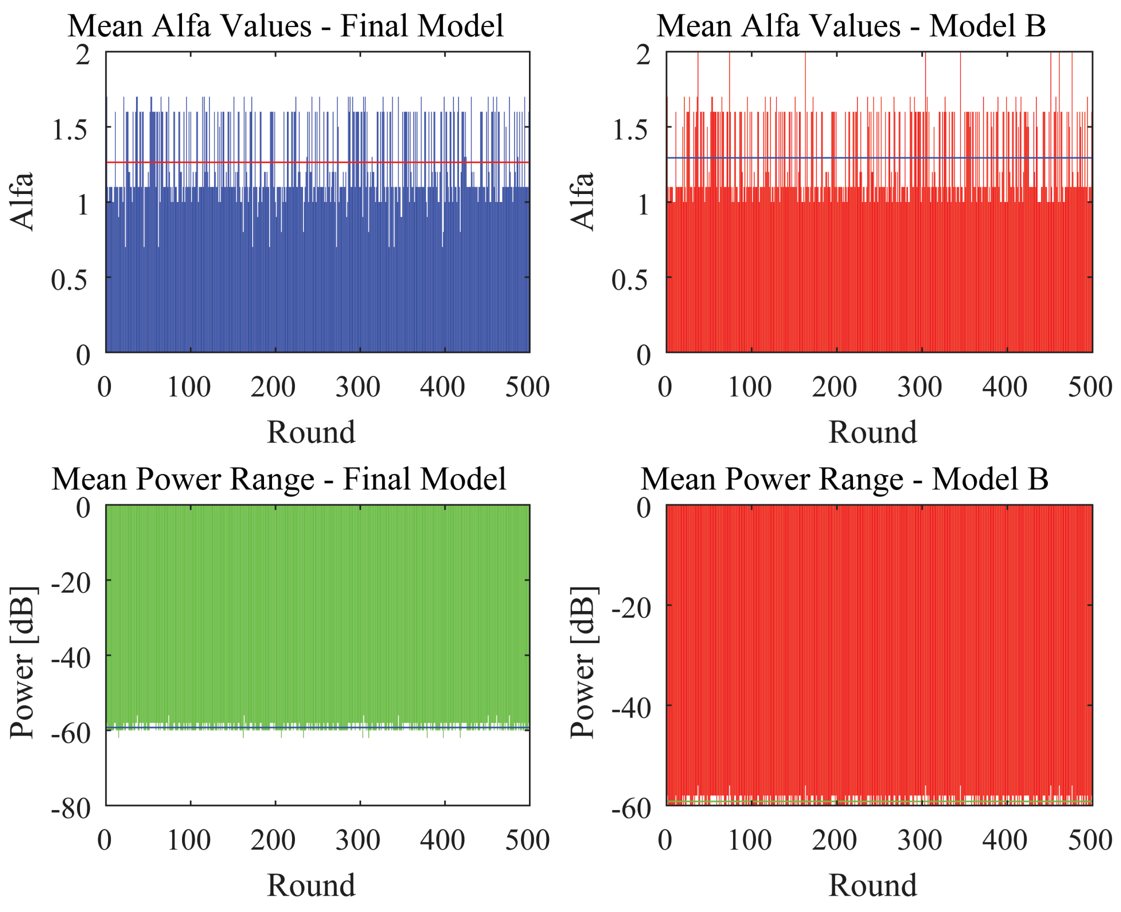


*Figure 5.16: Model's parameters confrontation - L.26.01 Classroom.*

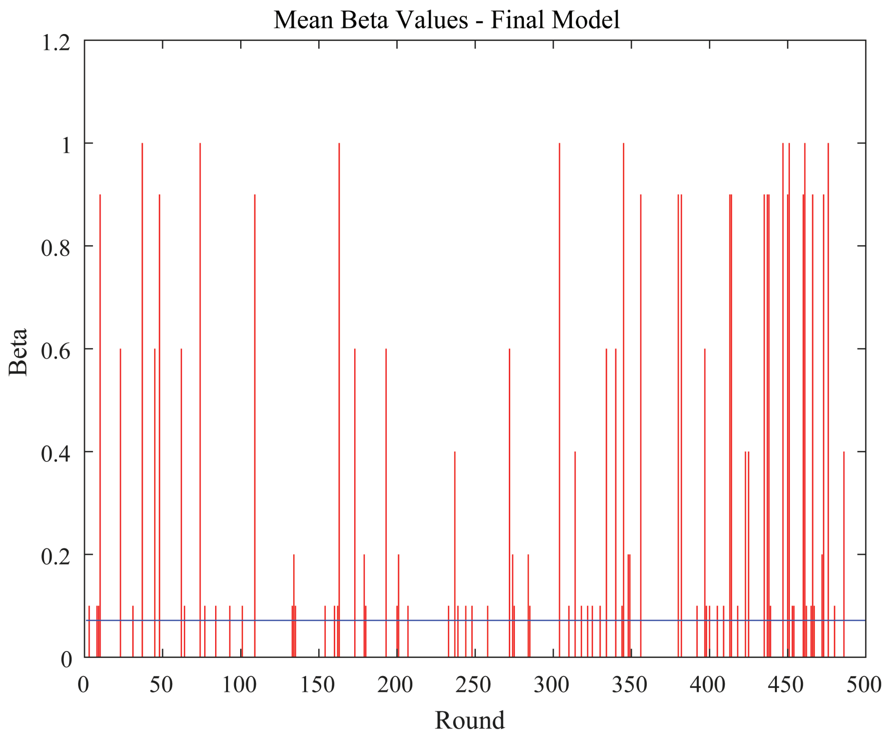


*Figure 5.17: The $\beta$ parameter variations with rounds, and its average - L.26.01 Classroom.*

### 5.3.5 Final Table

In this section, a final table is presented. This table summarize all the measurements of the previous section, related only to the final model. It is divided into 9 columns:

**Room** : the name of the room tested.

**Dim.** : the approximate dimension of the room in square meters.

**N° of seats** : the capacity of the room in number of seats available.

**Power Limit** : the range of power considered giving the minimum error. It's expressed in decibel (dB).

**RMSE** : the average error expressed in percentage related, this time, to the mean occupancy of the room.

**Av. Error (MAE)** : the average error committed by the model during its 500 rounds.

**Av. Error in %** : the average error expressed in percentage with respect to the total capacity of the room.

**Max Error** : the maximum error, in number of persons, encounter during all the rounds.

**Min Error** the minimum error committed by the model. Unless the model is really imprecise, this will be always 0.

| **Room** | **Dim. [m$^2$]** | **N° of Seats** | **Power Limit [dB]** | **RMSE** | **Av. Error** | **Av. Error [%]** | **Max Error** | **Min Error** |
|---|---|---|---|---|---|---|---|---|
| **ANTLab** | 100 | 21 | -79.45 | 2.14 | 2.23 | 10.6 | 15 | 0 |
| **Sala Tesisti** | 21 | 18 | -79.3 | 1.61 | 1.72 | 10.75 | 10 | 0 |
| **B.5.3** | 60 | 80 | -55.8 | 4.14 | 7.8 | 9.75 | 35 | 0 |
| **L.26.01** | 54 | 70 | -59.23 | 2.04 | 4.37 | 6.24 | 15 | 0 |

The results of these tests shows that there are some room for improvements. The average error related to the total rooms capacity always remain around 10%, that is a good value for a detection based on only one sensor's probe capture.
Another peculiar result is that, except for room B.5.3, the model B that doesn't consider the random MAC addresses performs slightly better. Although in the real device, an error of 1.8 or 2.2 doesn't change the results that will be always 2 (you cannot make a mistake of 2.2 person). Therefore in the real deployment the two models are similar. As

said in 4.1.3.2, we choose the Final model instead of the B one because, in our opinion, it's more sustainable in a world with growing random techniques implementation.
From the table we can see that the values of power used are different and seems to be independent from the dimension of the room. There is a simple explanation for this values: in crowded environments (such as those of the last two rooms analysed) the model prefer to keep the power lower, but increase the $\alpha$ and $\beta$ values. On the contrary in the first two rooms the parameters are kept lower, while the power range (the "radius" of sight of the chip) is larger. This could be again caused by the position of the rooms: the last couple of rooms are placed in more crowded buildings with respect to the first couple. Furthermore, of the last two rooms we have a smaller amount of data: the model is trained over 7 inputs against the 40 inputs of the first two rooms.
The minimum error is always 0 and it's correct as a value. It means that in some instants the estimation matches completely the real world observation. The maximum error although it's impressive at a first look, but consider that it's really rare and in real condition and could be the effect of a sudden change in the number of real people not immediately registered by the sensor. Let's clarify this last statement: the probe request capture has the problem of being slow on reacting to sudden changes. If, for example, a group of 30 students enters in a room, the sensor captures their probe request right away, with no problem. But if the same group exits rapidly from the room, the sensor doesn't react immediately but takes at most 7 minutes to remove inactive MAC addresses. In case like these the error can drastically increase until reaches the values in the table. The risk on taking a lower time to remove memorized MAC addresses is to remove devices that are still present in the room but not talkative (see [12]).

# Chapter 6

# Conclusions and future works

The last years have been characterized by a growing interest in smart buildings, smart objects and everything related to simplify everyday tasks and optimizing the resources used. In chapter 2 we've seen how occupancy detection can optimize public transportation, or power consumption in buildings. We have also seen the privacy risks related to a malicious use of probe requests' data, like the possibilities of tracking people. Finally we have also seen how modern OSs tries to use MAC address randomization in order to protect their users' privacy.
The main drawbacks of the techniques we've seen are the use of specific high cost hardware, powerful SoCs used for trivial sniffing tasks and the amount of space this hardware needs. In chapter 2 we've seen a large use of Raspberry Pi or other specific hardware like WiFI PineApple, that are all powerful devices around 35-70€. Although still affordable, these devices can be replaced by other low-cost hardware for the simple probe-capturing and analysis tasks. Despite the growing interests in this subject, as a part of the Smart-Buildings era, a very low-cost infrastructure for occupancy estimation has not been deployed yet.

## 6.1 Thesis achievements

At the end of this work of thesis, the goals achieved can be summarized as follows:

- Development of a low cost and small sensor (5cm x 4.5cm x 2cm) with Wi-fi capabilities and temperature, humidity, pressure, altitude and brightness sensing.
- Development of an estimation model to calculate the number of present persons using Wi-Fi probe detection.

- Development of the front-end part, with the communication with the sensor through MQTT protocol, and two methods of retrieve room's information: classical website method[1] and Telegram's bot alternative.
- Deployment of the sensor for in-site measurements.
- Analysis of the measurements for model's performance characterization.

**Sensor's development**
In chapter 4 the sensor's wiring and PCB development is discussed. For a smart sensor the dimensions and costs are important, and we decide to reduce them at a minimum. At the actual change, all the components used can be found in an online show at more or less 5€. To produce 10 of these sensor, considering using an external site for the PCB production[2], the price will be around 60€.

**The estimation model**
Chapter 2 subsection 4.1.3.2 describes the estimation model used. The results doesn't take into account the environment measurements but only the probe's count. The operation done by the chip is a minimization of the Root Mean Square Error calculated taking into account a maximum of 40 past measurements. The minimization function gives the correct parameters ($\alpha$, $\beta$ and Power) in order to calculate the room's occupancy.

**Front-End development**
In section 4.2 the server-side implementation is discussed. The work started with a "home-made" server based on a Raspberry Pi 3, and then moves the configuration onto Amazon Web Services. The installation and configuration of an MQTT and Node-RED server is performed, and then the flow implementation on Node-RED conclude this part of the project. With the flow implementation it was possible to create a dashboard site publicly reachable, and a Telegram's bot in order to give more choices for retrieving data at the final users.

**Sensor Measurements and model's performance evaluation**
In chapter 5 the model's performance in four different locations is evaluated. The first two locations were also an evaluation of the developed sensor, in order to "stress-test"

[1]www.poliaule.ml
[2]EasyEDA.com, for example, offers an online PCB making tool and shop to order them at low price.

its capabilities and improve some possible bugs[3]. The developed model gives an error around 10% with respect to the rooms total capabilities. In number of people, this error was always around 1-4 people, as it can be seen in table **??**.Using only probe requests as information about occupancy, and considering the limitations due to slow reactivity to rapid changes in the number of people present, these results are encouraging, and leave some space for improvements.

## 6.2 Future Works

This work has presented the development of a complete board with sniffing and measurements capabilities. There is some scope for further improvements, mainly driven by the experiences gained during the development. In this section some of those improvements are presented.

**Using temperature, humidity and pressure data to help the estimation process**

As we've seen in [8], temperature, humidity and other ambient measurements can be used in order to estimate the occupancy in an indoor environment. In the developed board there are already temperature, pressure and humidity sensors, but these values are not taken into account in the model. A future improvement will be to modify the model in order to keep track and help the estimation process using also these ambience data already collected.

**Using vectors of coefficients $\alpha$ and $\beta$**

In our model, after the training part, we use a single value of $\alpha$ and $\beta$ in order to calculate the estimation. A possible future work could be the evaluation of the feasibility of using coefficients vectors, instead that single coefficients, with a dimension equal to the number of power-range considered. The model will have to calculate every element of these vectors in order to minimize the total RMSE obtained. This evaluation will have to consider the computing power limitation of the low-cost chip.

**Optimizing the model's runtime**

In the deployed model we've seen (4.1.3.2) that the parameters' calculation requires a lot of time (almost 4 seconds). An improvement will be to find a way to reduce this calculation time reducing the number of operations done by the MCU.

[3]On this note, see the watchdog timer problem in section 3.4.2.1 and 4.1.3.2

### Using LPWAN[4] instead of Wi-Fi for data uploading

One of the limitation of the ESP8266 is that it doesn't support WPA/2 Enterprise Wi-Fi configuration with EAP-TLS authentication. The University network also blocks MQTT port for communication. For these reasons the chip cannot be deployed in all the rooms, but needs always a dedicated Wi-Fi network in order to send it's data to through MQTT to the server.

During the latest years, new chips were developed enabling the use of both Wi-Fi connectivity and a LPWAN technologies like LoRa[5]. Using powerful chips with these communication techniques, it will be possible to leave the sniffing mode always on and using the LoRa technology in order to send data to the gateway. This method will solve the Wi-Fi authentication limitation of ESP8266 and increase the computing power available.

[4]LPWAN is an achronym for Low-Power Wide Area Network. It's a type of wireless network designed to allow long range communication at low bit-rate for smart object. The low bit-rate is essential for sensors operating on a battery, in order to reduce power consumption.

[5]LoRa is a proprietary radio modulation technique base on spreading spectrum techniques. This wireless network operates in licensed-free frequency bands like 169 MHz, 433 MHz, 868 MHz and 915 MHz.

# Appendix A

In the figure below A.1 a signal and a random barker sequence are represented. It can be seen that the XOR operation (rules recap in table A.1, where A and B are the inputs and R the output) between the signal and the barker sequence, generates a signal with rate equal to that of the barker sequence, higher that the signal S(t). Given that for higher rate we have higher bandwidth requirements, the multiplication of the two signals generate the so-called "spreaded signal", a signal in which the power is spread along a larger interval of frequencies.

| **A** | **B** | **R** |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*Table A.1: XOR Rules*

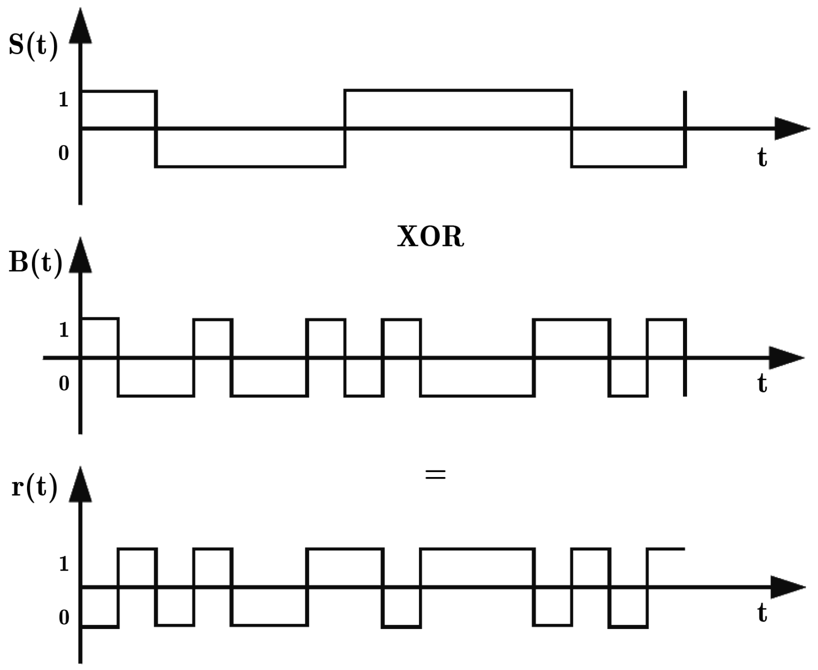


*Figure A.1: An example of signal and barker sequence multiplication*

# Bibliography

[1] V. Acuna et al. “Localization of WiFi Devices Using Probe Requests Captured at Unmanned Aerial Vehicles”. In: *IEEE Xplore* (2017).

[2] Jonathan Brooks et al. “An experimental investigation of occupancy-based energy-efficient control of commercial building indoor climate”. In: *IEEE Xplore* (2015).

[3] Jonathan Brooks et al. “Energy-efficient control of under-actuated HVAC zones in commercial buildings”. In: *Science Direct* (2015).

[4] Sartori Camilla. “Estimating users’ provenience through analysis of wi-fi probe requests”. Master Degree thesis. Politecnico di Milano, 2015–2016. URL: `http://hdl.handle.net/10589/132469`.

[5] IEEE. *IEEE Registered OUI.* 2016. URL: `https://standards.ieee.org/develop/regauth/oui/oui.txt`.

[6] ISO. *ISO - International Organization for Standardization.* 1946. URL: `www.iso.org`.

[7] Florian Dorfmeister Lorenz Schauer and Florian Wirth. “Analyzing Passive Wi-Fi Fingerprinting for Privacy -Preserving Indoor-Positioning”. In: *IEEE Xplore* (2016).

[8] Véronique Feldheim Luis M. Candanedo. “Accurate occupancy detection of an office room from light, temperature, humidity and CO2 measurements using statistical learning models”. In: *Science Direct* (2015).

[9] Célestin Matte et al. “Defeating MAC Address Randomization Through Timing Attacks”. In: *ACM WiSec 2016* (2016).

[10] OASIS. *OASIS - Advanced open standards for the information society.* 1993. URL: `www.oasis-open.org`.

[11] Thongtat Oransirikul et al. “Feasibility of analyzing Wi-Fi activity to estimate transit passenger population”. In: *IEEE Xplore* (2016).

[12] Julien Freudiger (PARC). “Short: How Talkative is your Mobile Device? An Experimental Study of Wi-Fi Probe Requests”. In: *WiSec ’15 Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks* Article No. 8 (2015).

[13] Achille Pattavina. *Reti di Telecomunicazione - Seconda Edizione*. McGraw-Hill, 2007.

[14] Simon Monk Paul Scherz. *Practical Electronics for Inventors - Third Edition*. McGraw-Hill, 2013.

[15] Weijun Qin et al. “Mo-Fi: Discovering Human Presence Activity with Smartphones Using Non-intrusive Wi-Fi Sniffers”. In: *IEEE Xplore* (2014).

[16] Waldemar Celes Roberto Ierusalimschy Luiz Henrique de Figueiredo. *Lua, an embeddable scripting language*. 1993. URL: `www.lua.org/`.

[17] Michael Moher Simon Haykin. *Communication Systems - Fifth Edition*. Wiley, 2010.

[18] Ars Technica. *No, this isn’t a scene from Minority Report. This trash can is stalking you*. 2013. URL: `https://arstechnica.com/information-technology/2013/08/no-this-isnt-a-scene-from-minority-report-this-trash-can-is-stalking-you/`.

[19] Edwin Vattapparamban et al. “Indoor Occupancy Tracking in Smart Buildings U sing Passive Sniffing of Probe Requests”. In: *IEEE Xplore* (2016).