POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree course in Communications and Computer Networks Engineering

Master Degree Thesis

WiFi Fingerprinting For Controlling Social Distancing

Supervisors
Prof. Paolo Giaccone
Prof. Claudio Ettore Casetti

Candidate
Ali Hojeij

Academic Year 2020-2021

Abstract

Internet of Things (IoT) has become a cutting-edge research topic in recent years to extend the massive internet connectivity to every physical device used in our daily life routine.

As the COVID-19 pandemic hits the smart cities like Turin, it is essential to manage the public transport boarding limitation to maintain the social distancing as much as possible. In this thesis work, the analysis of people's boarding on a bus is implemented by electromagnetic fingerprinting using a Raspberry PI 3 with a sniffer mounted on the bus, that is connected to the internet by a 4G SIM modem and connected to a network of the bus that provides essential information (e.g. door status, location, line ID, etc).

The sniffer captures WiFi probe request messages sent frequently by active devices searching for known access points. Each message contains a randomized MAC address that needs to be matched with other random MAC addresses corresponding to the same device for it to be counted as a unique passenger. Taking to account that not all people keep the WiFi interface enabled, or not all of them have a smart device, a correction factor and a smoothing process are applied. At each bus stop, our system returns an estimated number of passengers to be stored in a database and then visualizes it on a real-time dashboard.

Test results prove that our system is efficiently and accurately classifying the bus utilization based on a confusion matrix. Moreover, it is possible to identify frequent patterns of a specific line at different time slots of any given day. Also, it is relevant to see the effect of the COVID-19 lockdown regulations on such patterns.

Acknowledgements

I would like to express my deepest gratitude to my supervisors' professor Paolo Giaccone and professor Claudio Casetti for their endless support and valuable comments and guidance through this thesis work. I highly appreciate their valuable advice and suggestions to improve my work.

I would like to extend my gratitude to Marco Rapelli and Riccardo Rusca, the player makers of this project. Great things are not done by one person, they are done by a team of people.

Nevertheless, I am thankful to my family and friends for the unlimited support and inspiration to allow me to continue my journey.

Contents

Chapter 1

Introduction

As technology is growing, IoT is crossing the essentials towards an easier life. Smart cities already started deploying the new mobile communication generation 5G which can support plenty of use cases, mainly IoT applications. The goal behind IoT is to have devices that self-report in real-time, improving efficiency and bringing important information to the surface more quickly than a system depending on human intervention. IoT makes virtually everything "smart" by improving aspects of our life with the power of data collection. Citizens interact with the smart city through the use of their smart devices.

Turin is one of the important smart cities in Italy, which has a population of around one million, more or less, where citizens strongly depend on public transportation service offered by Gruppo Torinese Trasporti (GTT). Having the option of precisely checking the number of travelers addresses the most applicable variant for GTT since it gives a vital proportion of adequacy. Long and short-term planning contributes to efficient resource management and guarantees that vehicles operate in the appropriate place and time. Moreover, the COVID-19 pandemic is raising an urgent problem which is social distancing. Knowing how the vehicles are utilized by passengers is crucial to re-plan the timetables and number of busses running accordingly.

There are different applications to enable the automatic people on board counting systems, such as infrared sensors, video processing, or suspension sensors. These applications are quite expensive to be implemented on a large scale. Thus, the electromagnetic fingerprinting-based application is optimal in terms of cost where neither a further action is required from the passenger nor a specific application is needed to be installed on their devices. Thanks to the standardization of 802.11 WiFi protocol that allows us to build a system able to passively capture probe-request sent by devices asking to connect to a known access point. However, MAC addresses reported in these probe-requests are randomized for user privacy threats due to expected maltreatment like location tracking and profiling for devices being sniffed [10].

To counteract the MAC randomization, it is a must to match different MAC addresses detected to a single device to count it at once. Using a recursive scoring algorithm, we can investigate how probable that two or more probe-requests with different MAC address links to the same device. This algorithm depends on data reported in the probe-request messages that are not randomized such as packet sequence number, timestamp, and high throughput capabilities. Although the original MAC address will stay unknown since our system is out of any tracking or profiling purposes.

The objective of the thesis is to show how it is conceivable to use IoT to precisely detect the number of devices, which can be viewed as equivalent to the number of passengers on board. Somebody could contend that there is a jumble between the number of passengers and the number of devices possessed. This is without a doubt evident, but people are supposed to own a smart device taking to account that Turin is a smart city with a well-designed network infrastructure that extends the internet connectivity all over the city. In any case, it is necessary to calibrate the output of our system introducing a correction factor and a forecast smoothing based on previous outputs. Our system may have some limitations and considerations to overcome that are listed below:

- The system performance may be affected if bus stops are very close to each other, (e.g. not enough time window to capture probe-requests to be sent by all active devices on-board);
- The system can detect only people carrying smart devices with active WiFi interface in which no clear behavior of how often the Wifi interface is enabled, it can be compensated by a static or dynamic correction factor;
- Long traffic-stopping can affect the performance of the system due to the possibility of capturing data from passers-by, whereas, if the bus is moving, it is less probable to capture data from outside, even if, it will be a message with a very low signal quality that is in turn discarded based on appropriate filtering.

Chapter 2

WiFi Fingerprinting

Before building our system, deep research was held into the relevant technologies and approaches required to understand the structure of the WiFi probe requests and the MAC randomization applied depending on different device vendors.

2.1 MAC Address

A media access control address (MAC address) is a unique identifier assigned to a network interface controller (NIC). Every network interface hardware has a unique MAC address, which is a 48-bit hardware unique identifier. Institute of Electrical and Electronics Engineers (IEEE) assigns the first 3 bytes (24 bits) that are known as the Organizationally Unique Identifier (OUI), then device vendors assign the 24 most significant bits (NIC) under the condition that they use each address once. An example is shown in Figure 2.1.

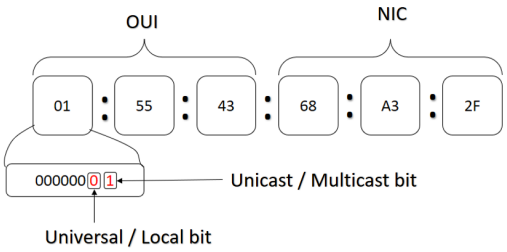

Figure 2.1. MAC Address Structure (reproduced from [13])

2.2 MAC Address Randomization

Wireless device users can be easily tracked if they are transmitting their unique identity (MAC address). So to ensure user privacy, main vendors have implemented MAC address randomization such that probe requests sent by a device don't use its real MAC address. There is no specific standardization for MAC address randomization in which each OS has its variant to implement the randomization algorithm. We recall that Apple supports MAC randomization starting from iOS 8 [10], whereas Android follows them by supporting MAC randomization starting from Android 6.0 [1].

According to [16], iOS apply MAC randomization every time the following events occur: **(i)** the device is locked or unlocked; **(ii)** WiFi interface is activated or deactivated; or **(iii)** a connection to a WiFi access point is made or attempted. Thus, it is not possible to estimate the time interval in which the same random MAC address is used. It strongly depends on the user's actions.

Researchers did an observation over a short time scale, recording that over one-hour, mobile devices send almost 55 probe-requests [10]. Another study ascertained that probe-request frames are sent in intervals of time ranging from 5 seconds to more than 10 minutes, with a tight density distribution of 88% of probe requests are sent during the first 5 minutes [12]. Unfortunately, whenever a new version of the mobile OS is released, the adequacy of these results may be disrupted.

2.3 WiFi Probe Request

WiFi-enabled devices can discover WiFi networks in two different ways: passive scanning or active scanning. This scanning process is done regardless of whether or not the user is connected to a wireless network, where the device sends a probe request messages even if a connection is established with an AP, to ensure the best signal quality if any AP is nearby with a better signal strength [14].

In passive scanning, stations (STA) listen on each frequency channel to the beacons to be sent by nearby AP which is not an efficient technique. While in active scanning, that most of the mobile vendors follow, STA sends to the broadcast address a *probe request management frame* asking what network is available on usable channels in an ordered way. Then STA starts a Probe Timer and waits for a possible answer from known AP, if no answers are received, STA moves to the next channel and repeats the scanning process.

We are required to analyze these frames which include information that is associated with a single device to achieve our goal. This information is the so-called Information Elements (IE) or the tagged parameters. According to [14], the tagged parameters can be used as a fingerprint to defeat the MAC randomization, in which they are not randomized by vendors as declared in [11].

As shown in Figure 2.2, a well know probe request structure is supported by 802.11 and we are interested in the following data:

- Source Address is the randomized MAC address of the STA;
- Packet Sequence Number, it is a part of *Seq-ctl* element, which is incremented for each transmitted frame (in cycle manner), with a value ranging from 0 to 4095 that reset to 0 when reaching its maximum value [7];
- The tagged Parameters are part of the *High Throughput (HT) Capabilities* that identify a family of devices but not the single device, which is a sub-element of the *frame body* element of the probe-request frame as shown in 2.3, that contains the following information:
  - HT Capabilities Information
  - HT Extended Capabilities
  - Transmit Beamforming Capabilities
  - ASEL Capabilities
  - A-MPDU Parameters

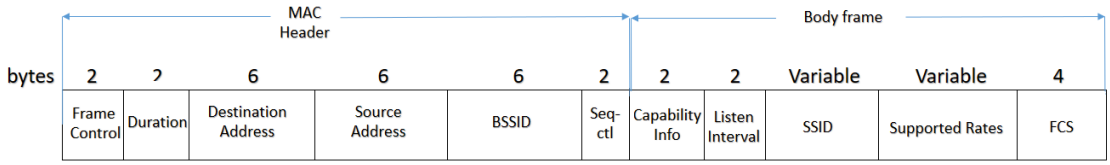

Figure 2.2. Basic Structure of 802.11 Probe Request Frame (reproduced from [11])

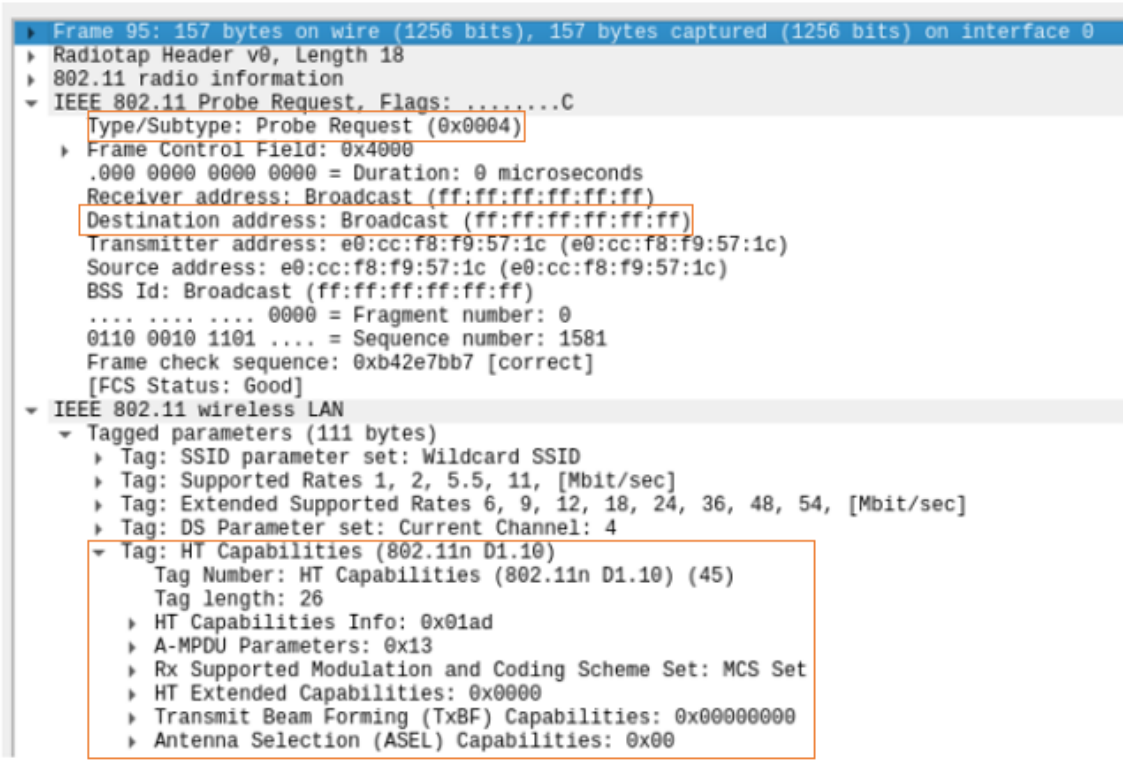

Figure 2.3. Wireshark Capture of Probe Request Frame Highlighting the HT Capabilities

Chapter 3

System Architecture

3.1 Choosing the hardware

For implementing our proposal on a large scale, the cost will be the main metric to consider, thus we seek the minimum cost possible with the best performance. For our project, dedicating an entire PC is overkill, thus a Single Board Computer (SBC) is the best option in terms of cost, power consumption, and space utilization. The hardware preferably must be a Linux-based operating system, for which a wide variety of tools is supported by Linux and can be easily configured by command line. We need a multiprocessor able to process data in to multitask manner, with RAM large enough to run the system, an LTE dongle to provide internet connectivity, an efficient WiFi USB-modem to capture probe requests, and a Network LAN port for plugging an Ethernet cable. Moreover, Taking to account that the available power on the bus supports a 5V/2.5A voltage/current output, we need hardware that copes with this specification.

There are plenty of options available in the market, such as Raspberry PI 3, Raspberry PI 4, Banana Pi, Odroid, etc. The prices range between $10 to over $250. A set of SBCs available in the market is reported with its specs in Table **??**.

Product	Price ($)	Processor	Cores	RAM	LAN	Wireless	USB ports	OSes
Raspberry Pi 3 Model B	35	Broadcom BCM2837	4 @ 1.2GHz	1GB	Fast	WiFi, BT	5	Linux
Banana Pi M2 Ultra	46	Allwinner R40	4 @ 1.4GHz	2GB	GbE	WiFi, BT	4	Linux, Android
Odroid-XU4	74	Samsung Exynos5422	4 @ 2GHz	2GB	GbE	opt.	3	Linux, Android

Table 3.1. Some SBC Specs in the market

According to [9], power and cost trade-off is done showing that Raspberry Pi 3B compromise as the best among the bench-marked SBCs. Thus the Raspberry Pi 3B is our final choice providing the specification we are looking for. We have tried to use Raspberry Pi 4 instead, trying to improve more the overall system performance

but the power provided by the bus system (5.1V/2.5A) is not compatible with the Raspberry PI 4 needs (5.1V/3A).

Since the built-in WiFi interface in the Raspberry PI 3 doesn't support the monitor mode [3], a USB WiFi dongle is used. This new interface must be configured as monitor mode, and the built-in device must be shut down to avoid any possible interference caused by it.

As what concerns WiFi antenna, we anticipate using a short-range WiFi antenna that covers the distance limited by the bus borders as much as possible. The higher the dBi of the antenna, the wider the coverage, and vice versa. An antenna with low dBi has 360 degrees coverage with tight distance. Thus we choose a commercial 2-dBi WiFi USB dongle to capture the WiFi probe requests.

According to the 802.11 standards, the typical transmission frequency bands for Wi-Fi are 2.4 and 5 GHz. The sniffer can capture data only at one frequency at a time, this would require two sniffers placed at the same point to capture on both bands. But processing data from two sniffers will increase the system complexity. Thus, the monitoring channel frequency was set to 2.4 GHz since all devices support 2.4 GHz, while old devices don't support 5 GHz. The main drawback is that 2.4GHz has fewer channel options with only three of them non-overlapping, while 5GHz has 23 non-overlapping channels.

We need a reliable server to store the system outcome data in a database. Thanks for Politecnico di Torino support, we have permission to use one of their powerful servers, namely *FULL00 Server*, that have a public static IP address. So we create a database running on the FULL00 server, see Chapter 3.5.5 for more information about the implemented database.

3.2 On-board System Description

With the collaboration of GTT, we mount a Raspberry PI 3 B on a bus that runs our system as shown in Figures 3.1 and 3.2. The Raspberry PI is power provided by the bus power system. It is connected to the bus network by an Ethernet cable, where the bus network broadcast the door status every one second using UDP transport protocol. The Raspberry PI is connected to the internet using an LTE module and plugged in with a WiFi USB dongle to capture the probe requests. The WiFi USB dongle is the default wireless interface of the system, where it is configured as a monitor mode. This configuration is done every time the system startup. Whereas, the capturing process starts right after the configuration, where the python chain scripts that apply the analysis and algorithm are executed a few seconds later.

3.2.1 Remote Accessing to Raspberry PI

The Raspberry PI is connected to the internet by LTE Network, where the IP address is dynamic and ISP firewalls don't allow the reception of the incoming SSH connections. Thus it is not possible to perform directly an SSH channel to the Raspberry PI. Instead, a persistent reverse SSH connection must be established upon system startup. Reverse SSH is a technique through which we can access systems that are behind a firewall from the outside world.

To establish a persistent reverse SSH, *autossh* tool is used. At each system startup, the */etc/rc.local* file is executed, in which a bash script is written inside this file. Before performing an autossh channel, it is required to create the Raspberry PI ssh-key, then copy it to the FULL00 server. This key is static, so it is done once and there is no need to create it at each system start-up. In this way, we can access remotely the Raspberry PI from the FULL00 server.

3.2.2 Collecting Data

Capturing WiFi Probe Request

The sniffer mounted on the bus is in charge of collecting the probe requests and saving them in a local file in real-time using *tshark* tool (part of Wireshark).

Using Wireshark we distinguish a probe request frame by its sub-type tag which is 0x04 as standardized by 802.11 as shown in Figure 2.3. With this support, we can report the timestamp and the received signal power of each captured probe-request frame. The data are stored in a local file *Captures.txt*, which in turn to be analyzed by the python scripts chain.

Capturing UDP packets sent by bus network

To read the broadcast data sent by the bus network, a UDP connection is established using *socket* library in python [8]. UDP does not require a long-lived connection, so setting up a UDP socket is simple. The IP address will be the broadcast address of the network (192.168.0.255), and the port will be any free port available. UDP messages must fit within a single packet and delivery is not guaranteed. Each UDP packet contains a set of information, such as door status, timestamp, Line ID, GPS location (latitude and longitude), speed of the bus, etc. These data are stored in a local file *Door.txt*, where every new entry received is appended to the file.

3.3 Python Chain Processes

The system performs a set of tasks, executed in a chain manner, where each task is performed in an independent python script that is in communication with other scripts by UNIX pipes. A UNIX pipe connects the STDOUT (standard output) file descriptor of the first process to the STDIN (standard input) of the second. When the first process writes to its STDOUT, that output can be immediately read (from STDIN) by the second process. In our case, we are using multiple pipes which are not different from using a single pipe. Each pipe is independent, and simply links the STDOUT and STDIN of the adjacent processes.

- **Time Window Sampling:** Read data from the local file *Captures.txt* that contain the information of probe-request frames being captured. Then seek for the last line and check its integrity (e.g. if the format is disrupted or the line is not complete). After that, perform a time window sampling according to door status fetched from local file *Door.txt* in such a way when the doors are opened, data captured are discarded, and the data collected so far will be constructed as an independent list L_1 to be analyzed in the following processes, then print the list L_1 to STDOUT;
- **Capture Analysis:** Read L_1 data from STDIN and analyze it. For each MAC address detected, a set of data must be reported in list L_2:
  - Number of occurrences of MAC address per list;
  - Received signal power average, maximum and minimum for each MAC per list;
  - Fist view and last view of each MAC per list;
  - First sequence number and last sequence number detected for each MAC per list;
  - OUI mapping for each MAC per list, if the first 3 bytes of the MAC is not found in the OUI list, the manufacturer is reported as "Unknown";
- **Capture Filtering:** Discard received packets in the list L_2 that have a MAC with an average received signal power less than a certain threshold (e.g. -70dBm) and the minimum number of MAC occurrences per list (e.g. 2). These values are in charge of analysis to be set (see Chapter 5). Then save the filtered list in L_3 and print it out to STDOUT;
- **De-randomizing Algorithm:** Read list L_3 from STDIN, then apply a recursive scoring algorithm to all 2-MAC combination in list L_3 that satisfy certain conditions (see 3.4.1). This algorithm assesses the probability that two or more random MAC addresses correspond to the same device. The

higher the score, the more probable the two MAC addresses relate to the same device. Save the result in list L_4 and print it out to STDOUT. A detailed description of the algorithm is mentioned in Section 3.4. A similar algorithm was built by [15];

- **Outcome Calibration and storing in database:** Read L_4 data from STDIN. The length of L_4 is the pre-estimated number of people boarding the bus. Then calibrate the result using simple exponential smoothing and adding a correction factor (these coefficients are in charge of analysis, see Chapter 5). Insert in the database the calibrated outcome, the instance timestamp, and the average load of the RP's CPU in the last one minute.

An overview of system architecture is shown in Figure 3.3.

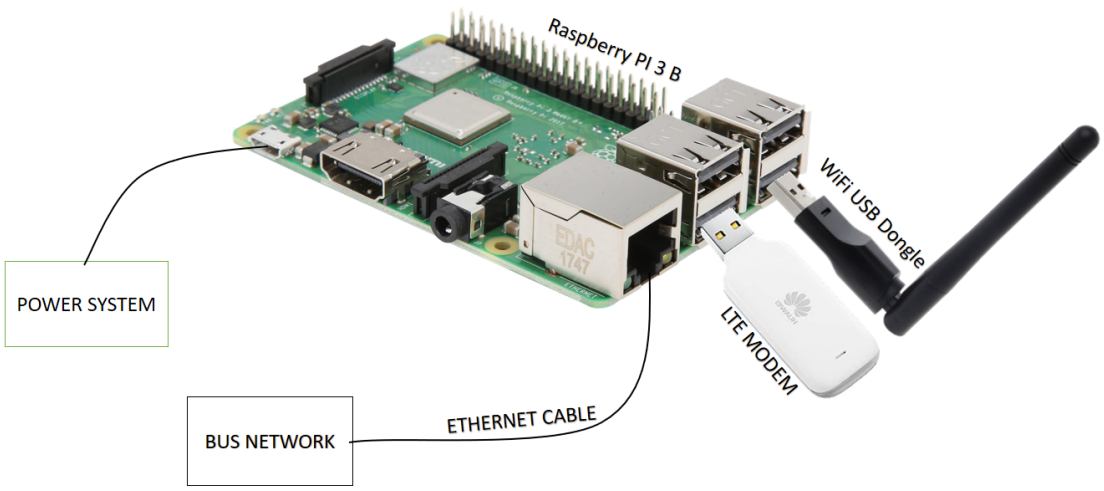

Figure 3.1. Raspberry PI 3 Plug-in Description

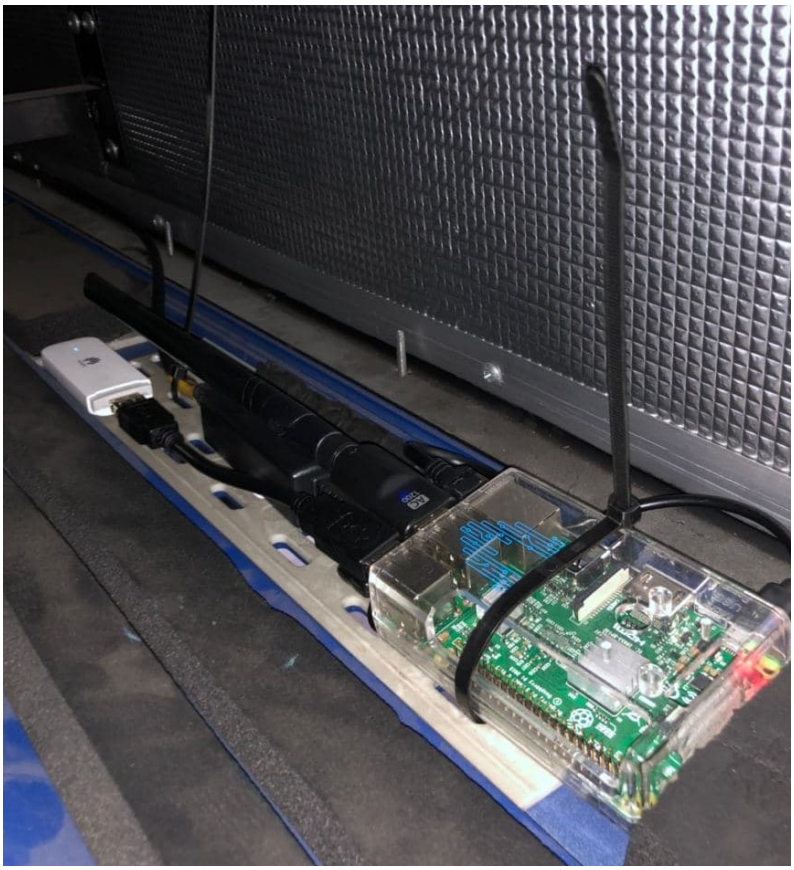

Figure 3.2. Raspberry PI 3 and the modules installed on the bus

Figure 3.3. System Architecture Flow Chart

3.4 De-randomization Algorithm

3.4.1 Conditions for Starting the Algorithm

As stated in Section 2.3, some elements included in probe-request frames can be exploited as a device fingerprint with a sufficient reliability. Elements such as sequence number and tagged parameters (HT Capabilities) of probe-request frames remain constant even with randomized MAC address. Considering two different MAC addresses captured by the sniffer: MAC_i and MAC_j, de-randomization algorithm starts if:

- **Condition 1**: First view of MAC_i $\mathbf{t_i^f}$ is before last view of MAC_j $\mathbf{t_j^l}$;
- **Condition 2**: HT capability (Tagged Parameters) of MAC_i is similar to HT capability of MAC_j.

3.4.2 Score Computation

We define a score for each 2-MAC combination available in a list. The score tells how probable these 2 MAC addresses relate to the same device, the higher the score the greater the probability. The score is the inverse proportion of:

- *[Δ TIME]*: The difference in time of the last view of MAC_i and the first view of MAC_j where it expresses the continuity in the arrival of the received frames;
- *[ΔSEQ]*: The difference in the sequence numbers between received frames and different MAC addresses. It expresses the continuity in the received frames, taking to account that the sequence number is cycling incremental value (range from 0 to 4095, then set to 0 when it reach its maximum value);
- *[ΔPOWER]*: The difference between the received signal power of each MAC. If the difference of received average signal power for the two MAC addresses is high, it is more probable that these two MAC don't relate to the same device.

Using the Equation 3.1 the score is computed:

$$Score_{ij} = \left| \frac{1}{\Delta TIME_{ij}} \cdot \frac{1}{\Delta SEQ_{ij}} \cdot \frac{1}{\Delta POWER_{ij}} \right| \tag{3.1}$$

where:

$$\Delta SEQ = \begin{cases} s_j^f - s_i^l, & \text{if } s_j^f > s_i^l \\ 4095 - (s_i^l - s_j^f) & \text{if } s_j^f < s_i^l \end{cases} \tag{3.2}$$

$$\Delta TIME_{ij} = t_j^f - t_i^l \tag{3.3}$$

$$\Delta POWER_{ij} = p_j - p_i \tag{3.4}$$

defining:

- s_j^f: Sequence Number of the first view of Mac_j
- s_i^l: Sequence Number of the last view of Mac_i
- t_j^f: Timestamp of the first view of Mac_j
- t_i^l: Timestamp of the last view of Mac_i
- p_i: Average received signal power of Mac_i
- p_j: Average received signal power of Mac_j

3.4.3 The Algorithm

Consider the dataframe D received from the filtering script that contains a set of MAC addresses and their associated information. Initially, the device list of list is empty. The first MAC in the dataframe will be appended to the first list of the device list. Each MAC in the dataframe will be assigned to a specific device list according to the following algorithm: If conditions 1 and 2 holds for at least one MAC in the list device at given index *ind*: save this index *ind* in the candidate list and set the found flag to 1. If the flag found not equal to 1, thus the $MAC_{current}$ certainly belongs to a new device, so the $MAC_{current}$ is appended to a new list of the device list, and the algorithm is stopped. Otherwise, compute the score between $MAC_{current}$ and all MAC inside the list with index *ind*. Then search for MAC_m having the maximum score computed.

Considering the list L_{ind} where MAC_m is located, if MAC_m is the last element of the list, $MAC_{current}$ is appended to the list, and the algorithm is stopped.

If there is another MAC address MAC_n that follows MAC_m in list L_{ind}, compare $Score_{m,current}$ and $Score_{m,n}$:

If $Score_{m,current}$ is less than $Score_{m,n}$, it means that it is more likely that MAC_m and MAC_n belong to the same device, with respect to MAC_m and $MAC_{current}$. Thus ignore MAC_m and recur for $MAC_{current}$ with an updated ignoring list. Ignoring means that the ignored MAC will not interfere again in the computation of scores when the algorithm recurs.

Otherwise, $Score_{m,current}$ is greater than $Score_{m,n}$, the MAC addresses following the MAC_m, are not sure to belong to same device. Thus trim them from the list L_{ind} and repeat the recursion for those trimmed MAC addresses to specify again the

more probable device they belong to. Then append the $\text{MAC}_{\text{current}}$ to the remaining list.

Applying this algorithm for all the available MAC in the dataframe D, the resulting will be a device list of list, that contains in each element of list of list the corresponding MAC addresses referring to the same device. Keep in mind that a limitation of recursion is set to stop the recursion in order not to reach the maximum recursion depth.

Summarizing, we state that the de-randomizing algorithm stops if: the recursion limit exceeded, the $\text{MAC}_{\text{current}}$ belong to a new device, or the MAC having the maximum score is the last element in the list. While it recurs if the MAC having the greatest score is not the last element in the corresponding list.

3.5 Pseudocodes for python chain scripts

3.5.1 Pseudocode of time window sampling script

```
1
2  flag=1 # it describes the preceding door status
3
4  while True:
5      read lastLine from local file Captures.txt
6      read doorStatus from local file Door.txt
7      check integrity for lastLine:
8          if doorStatus=0: # Door is closed
9              if flag == 0:
10                 list = list + lastLine
11                 flag = 0
12             else:
13                 list = lastLine
14                 flag = 0
15
16         else: # Door is opened
17             if flag == 0:
18                 print list to stdout
19                 list = ''
20                 flag = 1
21
22             else:
23                 lines = ''
24                 flag = 1
```

3.5.2 Pseudocode of Capture Analyser Script

```
1  list=[]
2  while True:
3      read dataframe D from stdin
4      calculate for each unique MAC in the dataframe D:
5          n =  Number of occurrence
6          ft = First timestamp
7          lt = Last timestamp
8          ar = average received signal power
9          fs = first sequence number
10         ls = last sequence number
11         Match the MAC with OUI list
12         if no match:
13             manufacture=unknown
14         append to a list the unique MAC and its corresponding data
15     print list to stdout
```

3.5.3 Pseudocode of Capture Filtering Script

```
1  L2=[]
2  while True:
3      read dataframe D from stdin
4      for each list L1 in D:
5          if L1.number_of_MAC_occurrence>=threshold_n:
6              if L1.average_signal_power>=threshold_s:
7                  L2.append(L1)
8      print L2 to stdout
```

3.5.4 pseudocode of the de-randomizing algorithm

```
1
2  # define D as dataframe that contain MAC addresses detected and its associated information
3  # read current_mac one by one from the dataframe D, and check it with all created list if any,
4  # define lst_device as a list of list,
5  # that stores, when recursion ends the MAC, addresses corresponding to each device
6  # define list_lst element list of lst_device
7
8  de-randomizer(current_mac, list_ignoring_mac, recursion_limit):
9      for j,list in lst_device:
10         for k,old_mac in lst_list:
11             if condition1(old_mac,current_mac) && condition2(old_mac,current_mac):
12                 lst_candidate.append[j]
13                 found=1
14                 break # whenever we found one MAC satisfiying the two coditions,
15                       # append the index of the list_device found in it to a tracking list and break
16
17     if found=0: # no MAC is satisfying the two equations,
18             # then current_mac certainly belong to new device
19
20         lst_device.append([current_mac]) # append to the list of list lst_device
21         #  the new MAC, and new list having a first element current_mac
22         list_ignoring_mac=[] # reset the ignoring list, since no matching MAC is possible yet
23         return # stop the randomizing algorithm
24     else:
25         for index in lst_candidate:
26             for  old_mac in lst_device[index]:
27                 if old_mac is not in list_ignoring_mac:
28                     if score>maxScore: #finding maximum score
29                         maxScore=score
30                         index_list=index
31
32                         #mac_candidate is the MAC with the highest score
33                         mac_candidate=old_mac
34                         max_score_index=lst_devices[index_list].index(mac_candidate)
35
36         if mac_candidate is the last element of the list:
37             # for sure mac_candidate is related to MAC addresses  inside the corresponding list
38             #thus append it to the corresponding list
39             lst_devices[index_list].append(current_mac)
40             lst_ignoring_mac=[] #
41             return # stop the algorithm
42
43         else: # the mac_candidate is not the last element,
44               #track the following_mac to mac_candidate
45               #the following score must be compared to max_score
46             following_score=computeScore(mac_candidate,following_mac)
47             if following_score > maxScore:
48                 lst_ignoring_mac.append(mac_candidate)
49                 # it is more likely that mac_candidate and following_mac belong to the same device
50                 # than mac_candidate and current_mac, thus ignore mac_candidate
51                 if recursion_limit>0:
52                     recursion_limit = recursion_limit - 1
53                     de-randomize(current_mac, lst_ignoring_mac, recursion_limit)
54                     # recurse again to find the right list for current_mac
55                 else:
56                     return # max recursion limit is reached
57             else:
58                 # the MAC addresses following the mac_candidate in the corresponding list
59                 # are not sure to belong to the same device anymore
60                 index_following = lst_list.index(following_mac)
61                 # save the mac addresses that are more likely not
62                 # belong to the current deviec, thus to be de-randomized again
63                 lst_repeating=lst_devices[index_list][index_following:]
64                 # save the mac addresses that are more likely belong to the same device
65                 lst_devices[index_list]=lst_devices[index_list][:index_following]
66                 # add current_mac to the end of the current list
67                 lst_devices[index_list].append(current_mac)
68                 # reset the ignoring mac list to pass it again in the following de-randomize recursion
69                 lst_ignoring_mac=[]
70
71                 for mac_repeating in lst_repeating:
72                     if recursion_limit>0:
73                         recursion_limit = recursion_limit - 1
74                         # recure again to find the right list for current_mac with updated ignoring list
75                         de-randomize(current_mac, lst_ignoring_mac, recursion_limit)
76                     else:
77                         return # max recursion limit is reached, stop the algorithm
```

3.5.5 pseudocode of Calibration and Sending to database

```
1
2  def exponential_smoothing(lst, alpha):
3      # F is the Forcast, lst is the data to be smoothed
4      # F[i+1]= alpha*A[i] + (1-alpha)*F[i]
5      F = []
6      F.append(lst[0])
7      for i in range(1, len(lst) + 1):
8          F.append(alpha * lst[i - 1] + (1 - alpha) * F[i - 1])
9      return F
10
11
12 while True:
13     establish connection to database db
14     if connection is established successfully:
15         while True:
16             read dataframe D from stdin
17                 num=len(D)
18                 buff.append(num)
19                 if(len(buff)>1):
20                     l=exponential_smoothing(buff,0.4)
21                 else:
22                     buff.append(num) #initial value for starting the smoothing
23             cpuLoad = average CPU load of last one minute
24             time = firstView of first MAC in dataframe D
25             detection = last element of list l
26             detection = detection + detection*correction_factor
27             query = insert_to_Table_in_database(time,detection,cpuLoad)
28             db.execute(query)
29             db.commit()
30     else:
31         sleep 5 minutes
```

Chapter 4

Database Storage and Data Visualization

A database is a collection of data stored in an orderly manner. To run a system efficiently and not lose any data from our system outcomes, the data must be stored in a database. These data can be fetched by a web server that runs the real-time dashboard. In this section, we will illustrate which type of database is suitable for our use case, and what are the options for visualizing the data on a real-time dashboard.

4.1 Types of databases

There are various types of databases. However, databases are widely divided into two major types, namely, Relational (Sequence Databases) and Non-relational (Non-sequence) databases.

4.1.1 Relational Database

A relational database is structured, meaning the data is organized in tables containing rows and columns. Each row is considered as a tuple or an entry, and each column stores a specific type of information. In other words, the column represents the data point itself that needs to be stored and the row is a record of the data points per column. The data within these tables have relationships with one another, or dependencies, where relationships are established through Primary and Foreign keys. Relationships can be easily defined between data points. Traditional relational databases are efficient at keeping the data transactions secure and making complex queries to acquire information.

Relational Database Advantages:

- Data is easily structured into categories;
- Data are consistent in input and easy to navigate;
- Can handle lots of complex queries, database transactions, and routine analysis of data;
- Easy configuration, simple import, and export of data;
- Ensure reliable database transactions.

Relational Database Disadvantages:

- Abundance of Information: Don't support storing very large images, numbers, designs, and multimedia products;
- Cost: Very expensive in setting up and maintaining the database
- Structured Limits: where relational databases have limits on field lengths. When you design the database, you have to specify the amount of data you can fit into a field.

4.1.2 Non-Relational Database

A non-relational database is document-oriented, meaning, all information gets stored in more of a laundry list order. Where NoSQL is less structured/confined in format, and thus, allows for more flexibility and adaptability. Non-Relational databases are efficient at storing large amounts of data semi-structured. Companies growing at a rapid pace like startups utilize more non-relational databases for their scalability and flexibility, where it can also save companies a lot of money.

4.2 Implemented Database

The database implemented is a relational database, because the data visualization tool we use supports only a time-series database that is a relational database. MySQL is a well-known relational database supported by the visualization tool we use, thus it is convenient to our use case.

Our MySQL database consist one table that have the following structure:

- ID [Primary Key], (int) auto-generated
- Timestamp, (datetime)
- Detection, (int)
- cpuLoad, (double)

The *Timestamp* is the instance when the bus stops (e.g. instance when the door is opened), the *Detection* is the estimated number of people at each bus stop, and *cpuLoad* is the average RP's CPU utilization in the last one minute.

4.3 Real-Time Visualization Tool

There are many tools available to create visualizations of large data sets. The best data visualization tools include Google Charts, Tableau, Grafana, ChartBlocks, etc. These tools offer a variety of visualization styles that are easy to use and can handle large data sets. Each one has its advantages and disadvantages. Among these options, we will try to identify the best visualization tool to be used. A comparison is held between Grafana and Google Charts

4.3.1 Grafana

Grafana is an open-source visualization tool that lets users create dynamic dashboards and other visualizations. It supports mixed data sources, annotations, and customize alert functions, and it can be extended via hundreds of available plugins. This makes it one of the most powerful visualization tools available. Grafana is advantageous in the following:

- Open source, with free and paid options available;
- Large selection of data sources available;
- Simple creation of dynamic dashboards;
- Built-in access control mechanism that allows restricted access to dashboards only to authorized users.

Nevertheless, Grafana has the following drawbacks:

- Very large database;
- Loading time depends on operation time on the database;
- Not the best option for creating visualization images;
- Not able to embed dashboards in websites, though possible for individual panels.

4.3.2 Google Charts

Google Charts is a powerful, free data visualization tool that is specifically for creating interactive charts for embedding online [5]. It works with dynamic data and the outputs are based on HTML5, so they work in browsers without the use of additional plugins. Google Charts have the following points of interest:

- Completely Free;
- Wide variety of chart formats available;
- Cross-browser compatible since it uses HTML5 customization options;
- Works with dynamic data;

All things considered, Google Charts have the following downsides:

- Beyond the tutorials and forum available, there's limited support
- Require a web developer to integrate it with their website

4.4 Implemented Dashboard

We decide to use the Grafana dashboard for visualizing the estimated number of people boarding the bus over time. Because Google Chart requires a website to be build which is not our point of interest. Instead, Grafana supports fetching data from the local MySQL database, where it can be accessed online using the static public IP of the server on a free port that is chosen upon the configuration of the dashboard.

We set two line charts, first for visualizing the estimated number of people boarding per bus stop as a function of time, and the other for visualizing the average CPU load evolution over time.

All the charts are interactive, so it is possible to change the time axis and zoom in a specific time interval to see more in detail all the data. Also, it is possible to set or modify the refresh rate of the data to see real-time values coming from the system.

Chapter 5

Experimental Evaluation

This chapter will show the analysis of captures received by our sniffer. We did six manual counting sets, each one with a different day and time. Based on the ground truth results, we set the best filter combination (i.e. received average signal power threshold for each MAC address and the number of the occurrence of a MAC per list), smoothing coefficient (α), and correction factor.

Moreover, we tested the performance of the de-randomization algorithm along with the analysis. It is also conceivable to identify a frequent pattern of people boarding the bus at different times of the day and during vacation and working days.

The manual counting is done on the following days:

- JULY 26th 2020;
- JULY 27th 2020;
- OCTOBER 16th 2020;
- OCTOBER 19th 2020;
- OCTOBER 30th 2020;
- NOVEMBER 2nd 2020.

During the lockdown, it is not possible to do any manual counting due to the strict regulations, thus we are limited with these sets only. Our system is mounted on bus ID 33E that is operated by GTT. It usually runs on specific lines as shown in table 5.1.

The lines with urban mobility type are usually more utilized by passengers, thus different patterns are expected for different mobility types.

Line Number	Path Figure	Mobility Type	Starting Region	Returning Region
6	5.1	Urban	Porta Nuova	Madonna Del Pilone
19	5.2	Urban	Porta Susa	Madonna Del Pilone
SE1	5.3	Sub-Urban	Torino Stura	Settimo Torinese
SE2	5.3	Sub-Urban	Torino Stura	Settimo Torinese
61	5.4	Sub-Urban	Porta Nuova	San Mauro Torinese

Table 5.1. Lines that bus 33E runs on

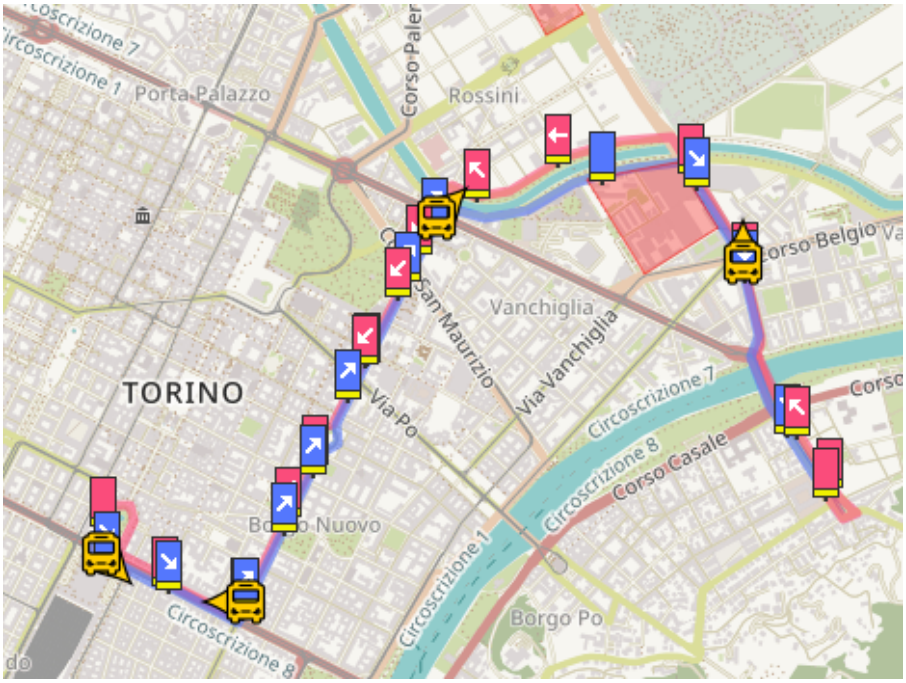

Figure 5.1. Line 6 Path

Figure 5.2. Line 19 Path

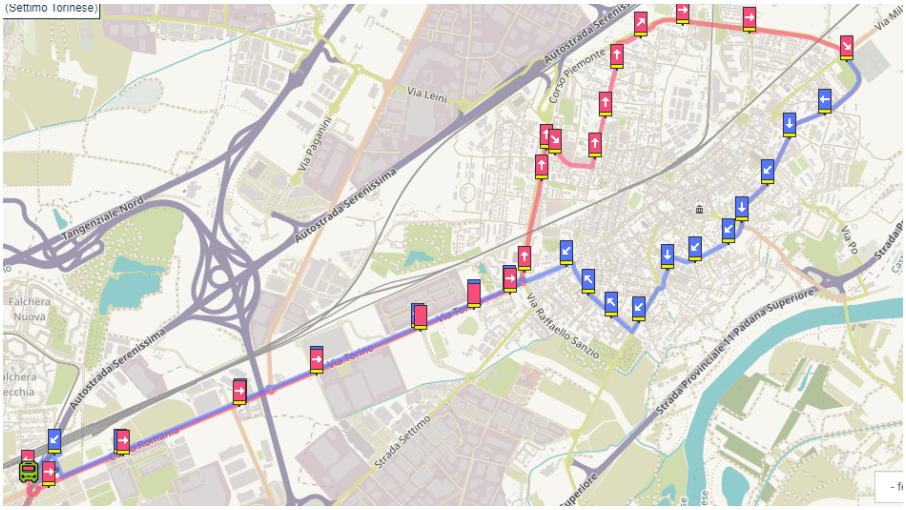

Figure 5.3. Line SE1 and SE2 Paths

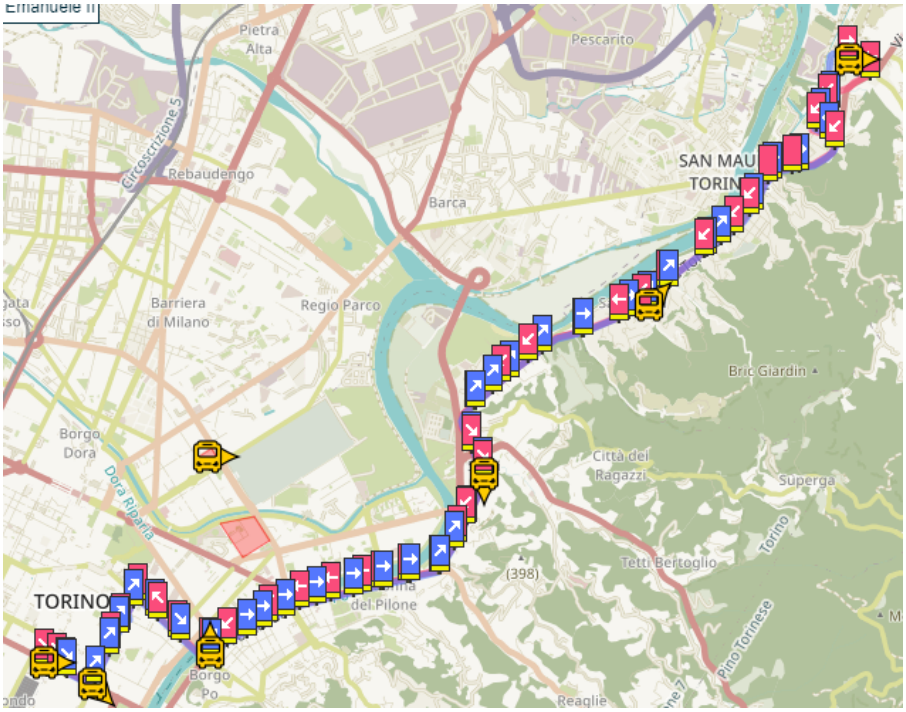

Figure 5.4. Line 61 Path

5.1 Profiling

Profiling is exploited to test the capability of our system. Since our system is build using python, which is high-level programming, we have to monitor the CPU load and the Memory Load to see if the system is capable to execute all the scripts simultaneously without overloading. In order to do so, *top* LINUX tool is used. *top* command provides a dynamic real-time view of the running system. This command shows the summary information of the system and the list of processes that are currently managed by the Linux Kernel. A bash script is written to monitor all the python scripts by their process ID (PID) using *top* over a time interval.

Fortunately, our system is under-loaded, with a maximum CPU load spike of 55% as shown in the Figure 5.10 and a stable memory load around 19.8% as shown in Table 5.2. The system is experiencing spikes in CPU load since the probe requests are burst-oriented so the processor will process a burst at once, which will significantly increase the CPU utilization at a single instance. Each python script of the chain has a different CPU load. Figure 5.5 shows the time window sampling CPU load, which is the least consuming with a max spike equal to 0.75%. Because no dataframe is created in this script, and the script is simple. (see pseudo-codes in Section 3.5). While capture analyzer, capture filter, and de-randomizing algorithm script have a much higher CPU load since they are more complex because data-frames are used and complex calculations are done in these scripts. The de-randomizing algorithm is the most CPU-consuming since it is the most complex among all the scripts with a maximum CPU spike of 22%. The process-related to calibrating the outcome and inserting it in the database have a different CPU utilization behavior, as shown in Figure 5.9. Since the connection to the database is frequently checked to not lose any data, thus the CPU load of this script is always fluctuating.

Process	Memory Load [%]
Time Window Sampling	0.7
Capture Analyzer	4.7
Capture Filter	4.5
De-randomizing Algorithm	4.5
Calibrating and Sending Outcome to DB	4.7
Total Memory Load	19.8

Table 5.2. Static Memory Load of Each Process

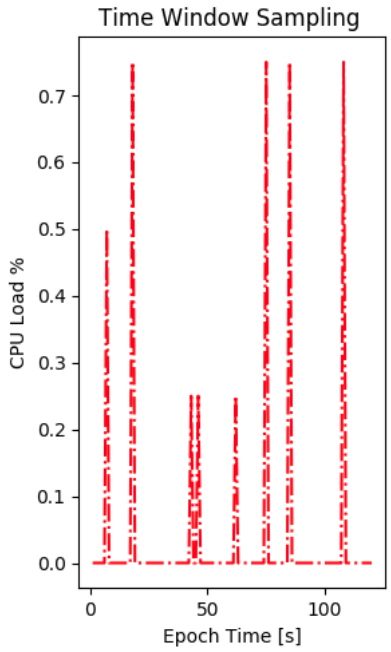

Figure 5.5. Time Window Sampling Script CPU Load Evolution Over Time

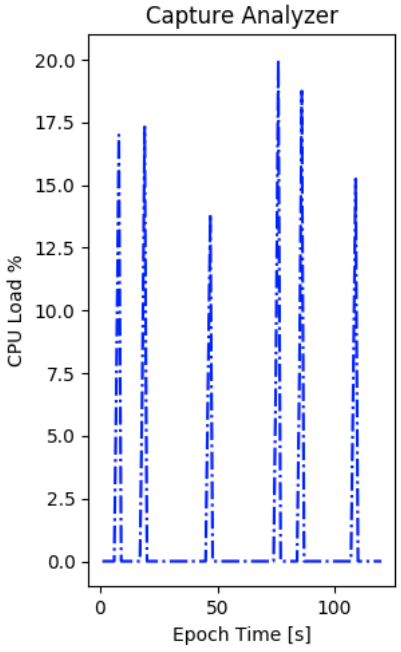

Figure 5.6. Capture Analyzer Script CPU Load Evolution Over Time

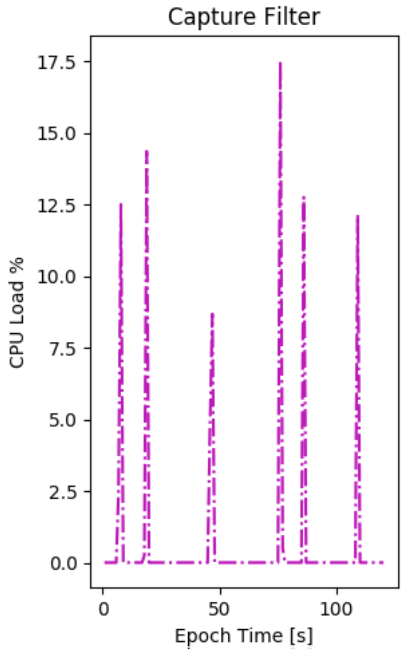

Figure 5.7. Capture Filter Script CPU Load Evolution Over Time

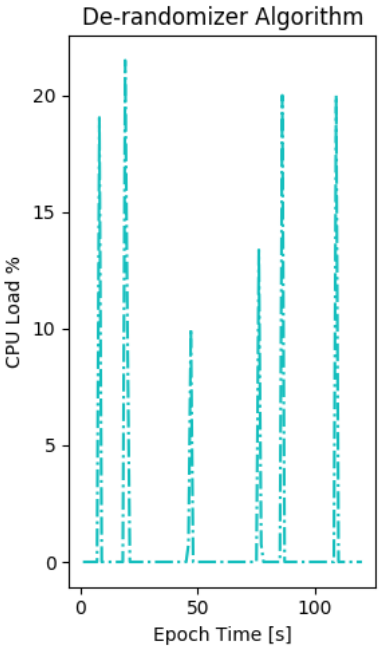

Figure 5.8. De-randomization Algorithm Script CPU Load Evolution Over Time

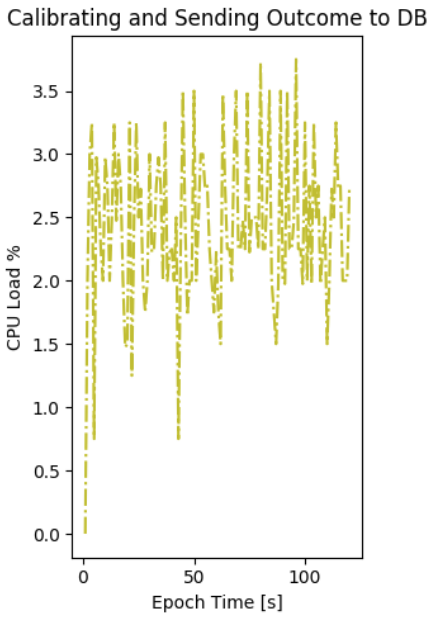

Figure 5.9. Outcome calibration and Inserting it in Database Script CPU Load Evolution Over Time

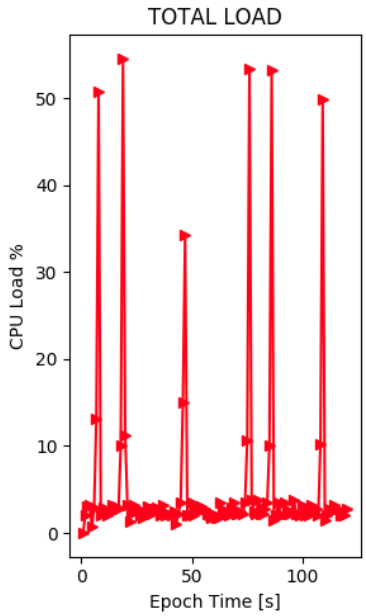

Figure 5.10. Total CPU Load Evolution Over Time

5.2 MAC De-randomization Algorithm Performance

In this section, we tested the performance of the de-randomizing algorithm by comparing the ground truth with the system outcome in two scenarios: with applying the derandomizing algorithm and without applying it. To have a fair comparison, the same filter combination was followed in both scenarios, in such a way that the signal power threshold is equal to -75dBm and the minimum number of MAC occurrence per list is 1. This algorithm is substantial since as illustrated previously, the same device sends probe-request with different randomized MAC address every time, so de-randomization is required to guarantee a single time counting of it.

The longer the sampling time window, the higher the number of times a device changes its MAC address and then the higher the number of distinct MAC address captured. The effect of derandomization is to reduce the number of MAC addresses received by a factor. The factor of reducing the number of MAC addresses is dynamic, which depends on the time window sampling and user's habit.

Taking to account that the MAC randomization strongly depends on the user's habit, where probe requests are sent upon special events done by the user. Each device applies MAC randomization after sending a probe-request burst. Hence, the factor of reducing the number of distinct MAC is dynamic (i.e. matching MAC addresses in a dataframe to the same device).

Moreover, If the sampling time window is tight, each device may send only one probe request burst having a unique MAC during the time interval, thus the reducing factor is equal to 1. Otherwise, each device has enough time interval to send one or more probe request burst, each time with a different MAC address. Thus the reducing factor will be greater than 1. Reducing factor equal to 1 means that each MAC address captured is not related to any MAC address in the dataframe (e.g. all scores returned by the derandomization algorithm are zero).

We can see clearly from Figure 5.14 that the maximum reducing factor is 5, meaning that on average each 5 distinct MAC detected is brought back to a single device. Whereas the minimum reducing factor is 1, meaning that each distinct MAC is detected as a single device. The same scenario is shown in 5.12, where the maximum reducing factor equal to 5.6, where the lowest reducing factor equal to 1. Also, in Figure 5.16 the minimum reducing factor is 1, but with maximum equal to 3.s

The comparisons are shown in Figures 5.11 , 5.13 and 5.15. The plots prove the efficiency of our derandomizing algorithm. When no MAC derandomizing was applied, the system outcome is relatively not reasonable (order of hundred distinct MAC detected). Whereas after applying the de-randomization algorithm, the outcome converges to reasonable values close enough to the ground truth values with some spikes to be smoothed later on.

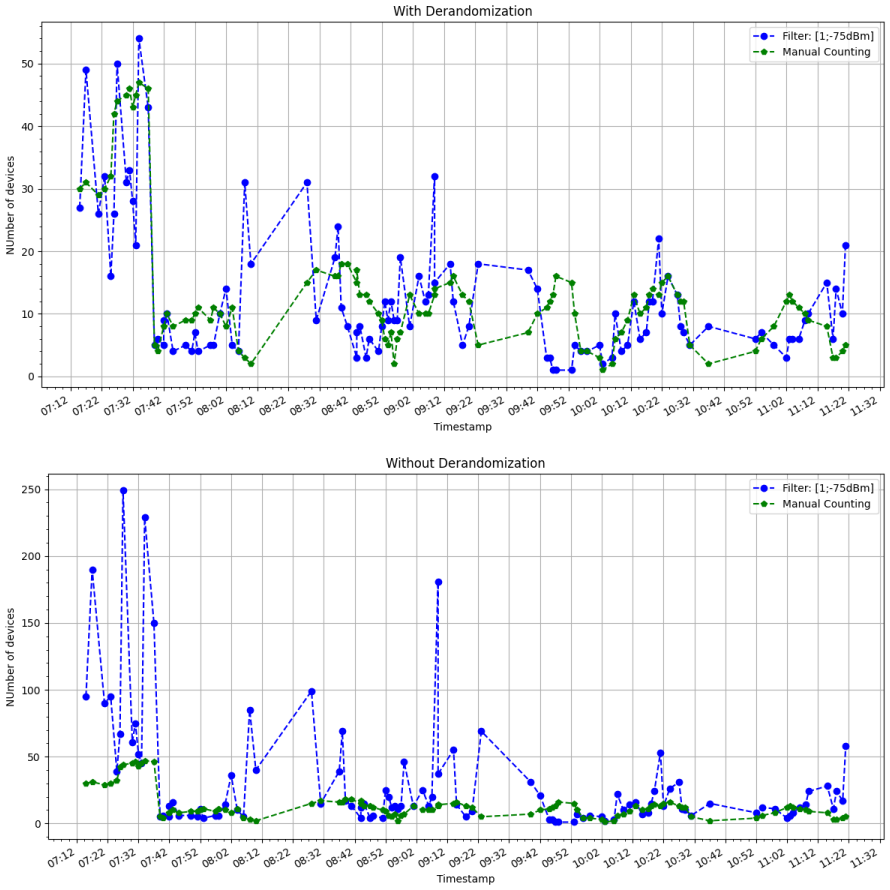

Figure 5.11. OCTOBER 16 - With or Without MAC De-randomization Comparison

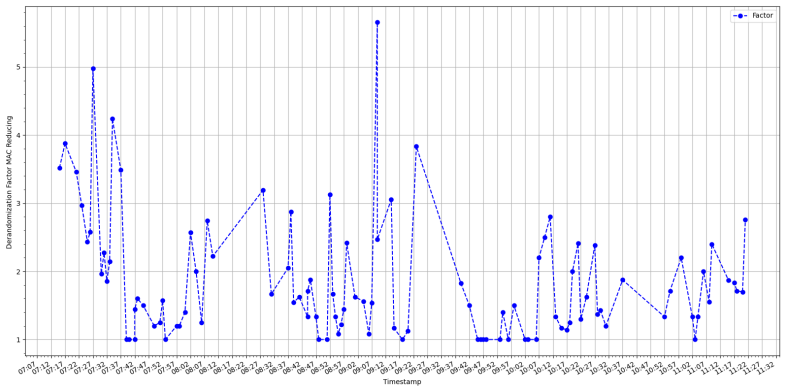

Figure 5.12. OCTOBER 16 - Factor of reducing number of MAC addresses Evolution over time

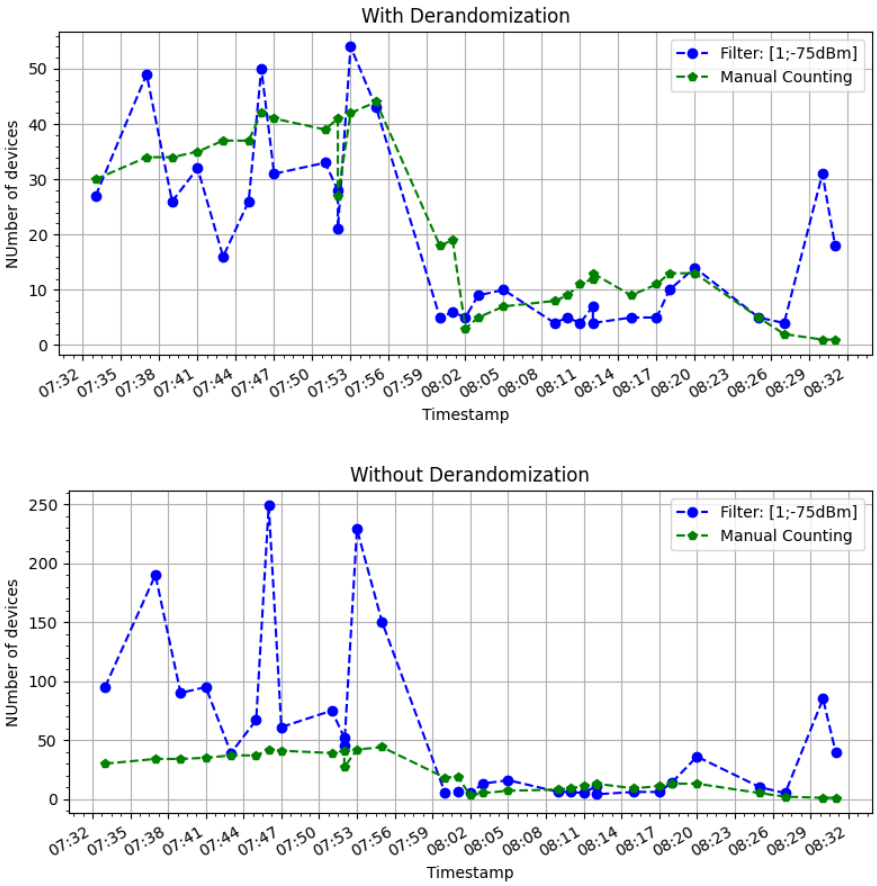

Figure 5.13. OCTOBER 19 - With or Without MAC De-randomization Comparison

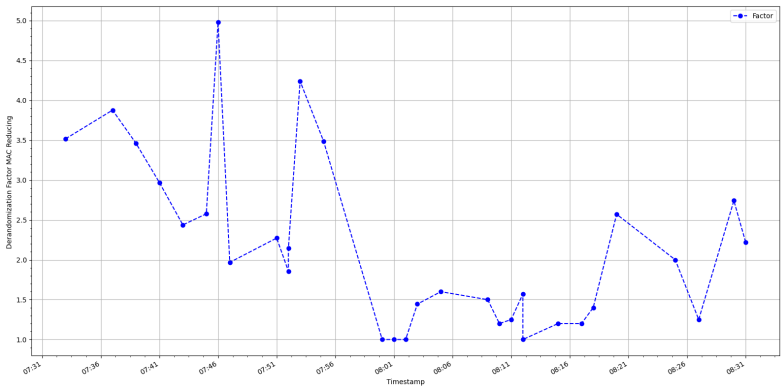

Figure 5.14. OCTOBER 19 - Factor of reducing number of MAC addresses Evolution over time

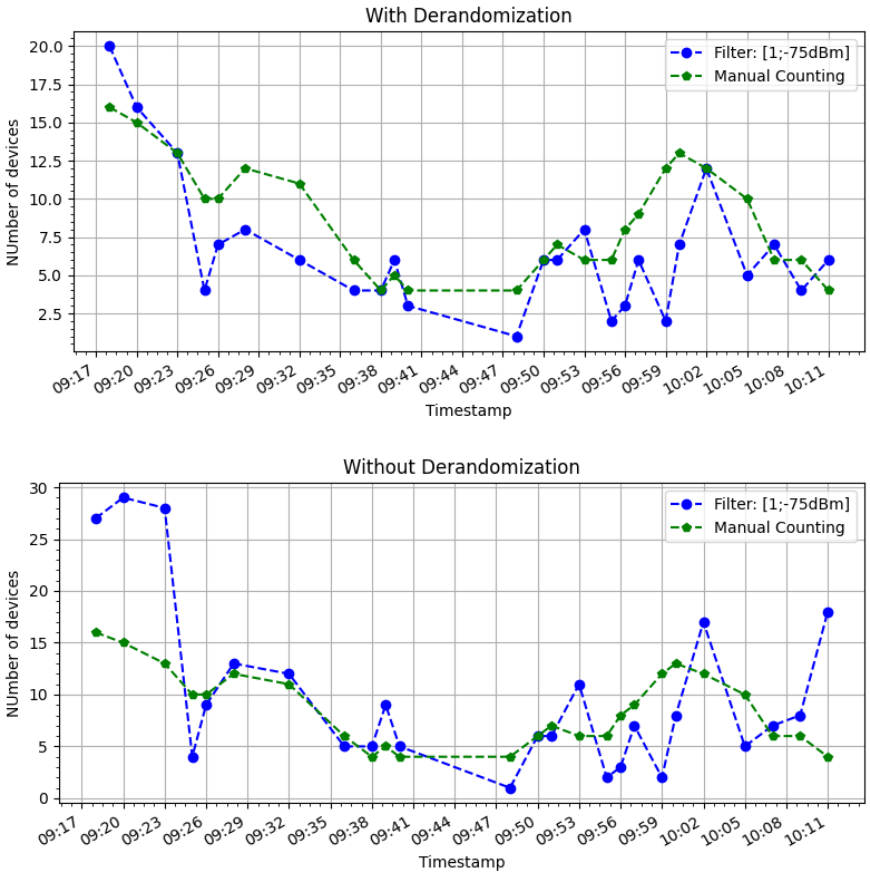

Figure 5.15. OCTOBER 30 - With or Without MAC De-randomization Comparison

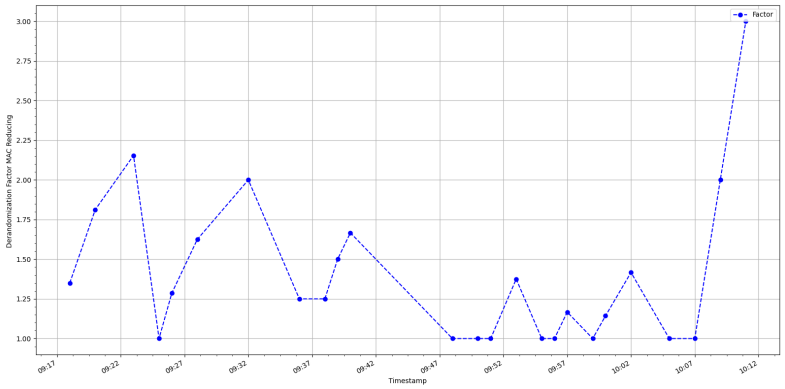

Figure 5.16. OCTOBER 30 - Factor of reducing number of MAC Addresses Evolution Over Time

5.3 Tuning System Parameters

5.3.1 Determining Optimal Filter Combination

The system parameters are tuned based on minimizing the absolute average relative error between ground truth and system detection. We test the system changing every time the MAC occurrence from 1 to 5, and the average received signal power threshold from -60dBm to -110dBm to find the best filter combination for each manual counting set. Taking into consideration the strong correlation between different manual counting sets, thus the best filter combination for each set will be almost similar.

It is clearly shown from the plots in Figure 5.17, the best filter combination is: minimum occurrence of MAC per list is equal to 1, and signal power threshold is -75dBm.

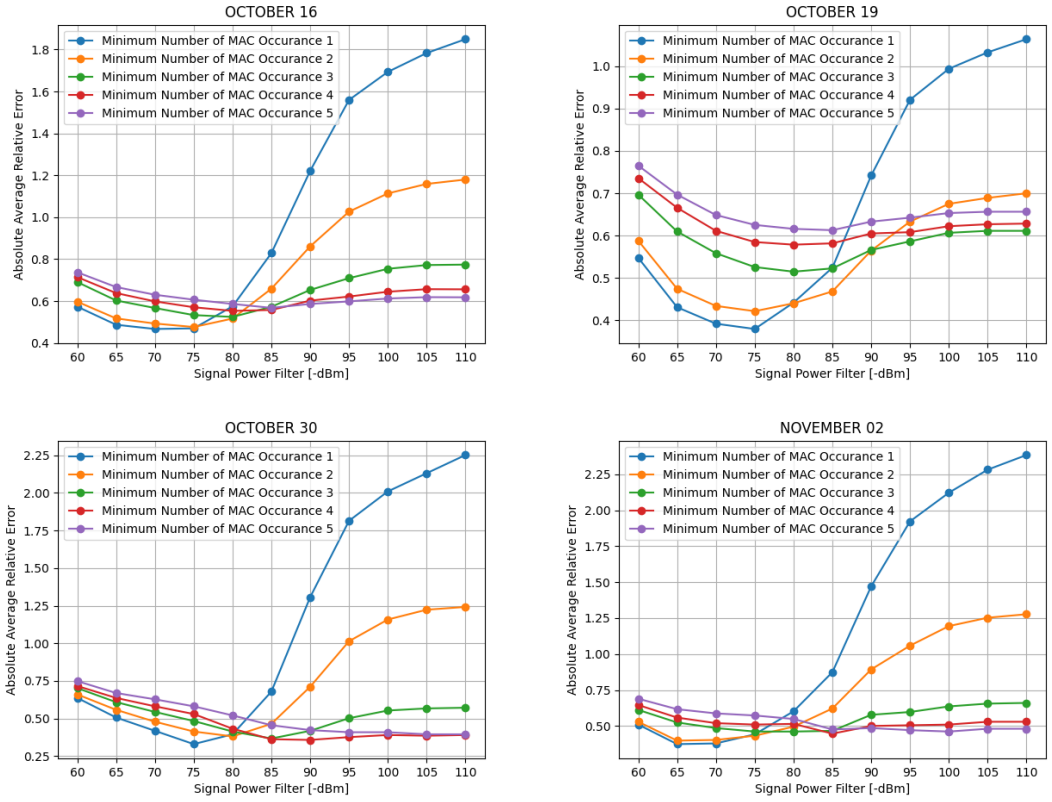

Figure 5.17. Plots for different Manual Counting to find the best filter combination based on minimizing AARE

5.3.2 Applying Smoothing

To overcome noisy measurements like unexpected spikes in estimating the number of people boarding the bus, it's essential to apply a smoothing process to our data based on previous measurements. There are different approaches to apply to smooth, such as *Simple Exponential Smoothing*.

Simple Exponential Smoothing is a time series forecasting method for one-dimensional data without a trend or seasonality. It requires a single parameter, called α, also called the smoothing factor or smoothing coefficient. This parameter controls the rate at which the influence of the observations at prior time steps decays exponentially. Simple exponential smoothing is described by the following equation:

$$F_{t+1} = \alpha \cdot D_t + (1 - \alpha) \cdot F_t \tag{5.1}$$

defining:

- The forecast F_{t+1} for the upcoming period is the estimate of the average level at the end of period t, in other words, F_{t+1} is a weighted average of all previous demand;
- D_t is the current value to be smoothed;
- F_t is the previous smoothed value;
- α: the smoothing parameter that defines the weighting and should be greater than 0 and less than 1. α equal 0 sets the current smoothed point to the previous smoothed value and α equal 1 sets the current smoothed point to the current point (i.e., the smoothed series is the original series). The closer α is to 1, the less the prior data points enter into the smooth.

Since the probe requests are burst-oriented, so the sniffer has to capture packets in a long enough time. If the time elapsed between consecutive stops is tight, the detection will be sharply decreased concerning the previous detection. On the other hand, the detection may have a sharp increase, since at crowd areas like Porta Nouva or Porta Susa, a lot of passers-by will be there and it is possible to capture probe requests from them. Thus we are obliged to use forecast average smoothing based on the preceding detection to smooth such spikes.

5.3.3 Choosing the Smoothing Coefficient

According to the function 5.1, values of α near one put almost all weight on the most recent observations, where Values of α near-zero allow the distant past observations to have a large influence.

The smoothing constant determine the sensitivity of forecasts to changes in demand. α **may be chosen either subjectively or objectively.**

Selecting the smoothing constant *subjectively*, depends on testing experience. As such if the mechanism generating the series has gone through some fundamental changes, choose a smoothing constant value of 0.9 which will cause distant observations to be ignored. While if the series is fairly stable and only going through random fluctuations, use a value of 0.1.

Selecting a smoothing factor *objectively* instead is better (which the way we will follow) based on some metrics in terms of error, such as *MAE (mean absolute error), MSE (mean square error), AARE (absolute average relative percentage error).* These errors are calculated concerning ground truth values to be measured (e.g. manual counting of people boarding the bus in our case). One of these metrics is followed to get its minimum value to find the most suitable smoothing factor for our case. Using this formulation, we can define the three error-size criteria as follows:

$$MSE = \frac{1}{n} \cdot \sum_{t=1}^{n} (A_t - X_t)^2 \tag{5.2}$$

$$MAE = \frac{1}{n} \cdot \sum_{t=1}^{n} |A_t - X_t| \tag{5.3}$$

$$AARE = \left| \frac{A_t - X_t}{X_t} \right| \tag{5.4}$$

defining:

- A_t: Ground truth value at period t
- X_t: Estimated value at period t
- n: number of observation per test

5.3.4 Initial Value For Exponential Smoothing

As noticed in Equation 5.1, the smoothing process needs an initial value F_1. But how can we set it?

A poor decision of the initial value will require a lot of time before the recursive smoothing formula to adjust. The effect is truly unmistakable among small-time series. In this manner, having an appropriate initial value will enhance the smoothing

process. There are different approaches for setting the initial value: **i** First Observation Value, **(ii)** Average value of the 1st few observations, **(iii)** Back-casting, **(iv)** Optimization.

Back-casting and Optimization techniques can't be implemented in the real-time scenario, since the series must be pre-defined. Taking to account that the probe request transmitting is burst-oriented, then maybe the first few detections of the system will not be adequate, where the impact of one observation value on the forecast value is mitigated. So no worth having an initial value equal to the average of the first few detections, therefore the initial value we set is equal to the first observation value.

By changing to the smoothing coefficient the absolute average relative error will be improved or getting worse by applying exponential smoothing. We test the smoothing coefficient (α) ranging from 0.0 to 1.0. The best smoothing coefficient detected is 0.4 by minimizing the absolute average relative error. The overall adequacy of the estimated number of people on the bus is improved as shown in Figure 5.18. Whereas the spikes are smoothed as shown in Figure 5.19 and 5.20.

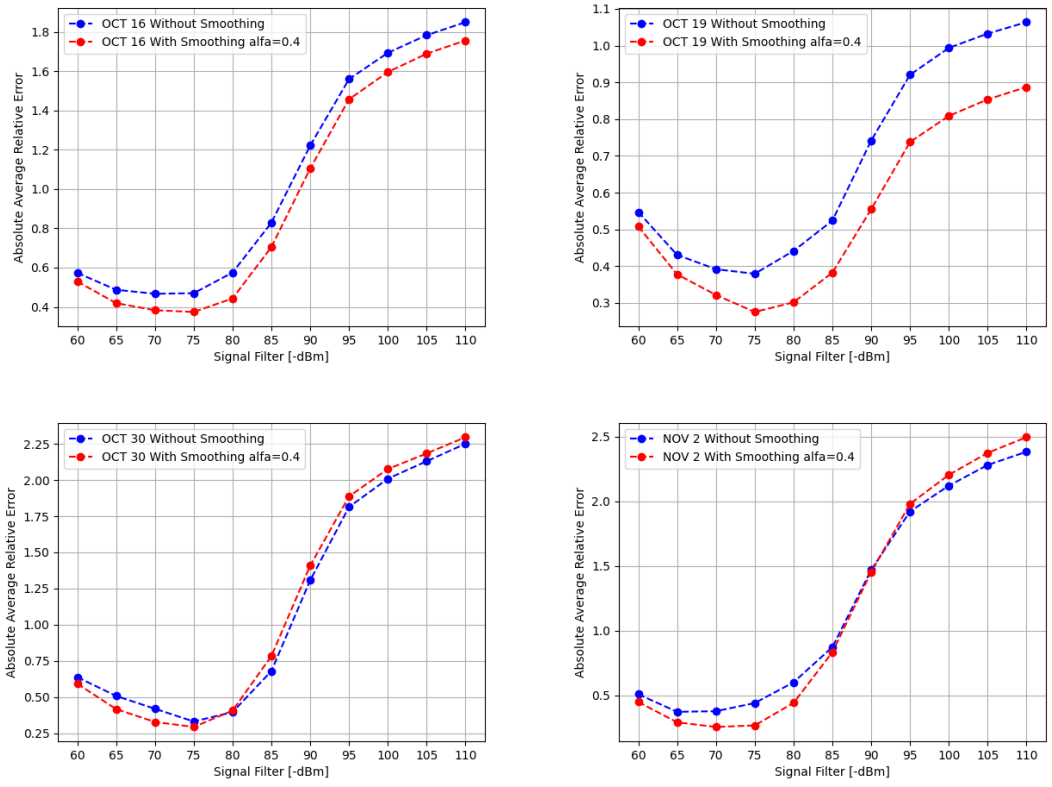

Figure 5.18. Comparison of AARE with and without smoothing for different manual counting sets with minimum number of MAC occurrence = 1

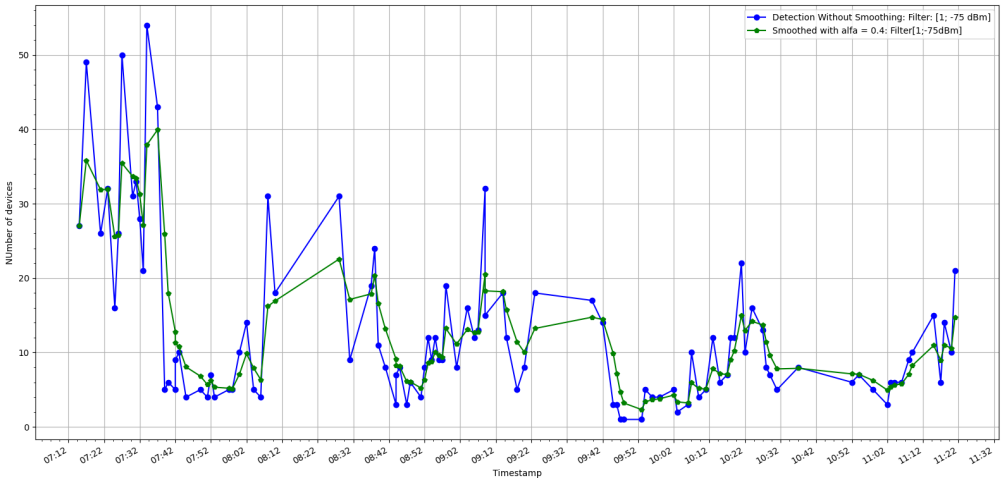

Figure 5.19. Spikes Smoothing $\alpha = 0.4$ - OCTOBER 16 SET

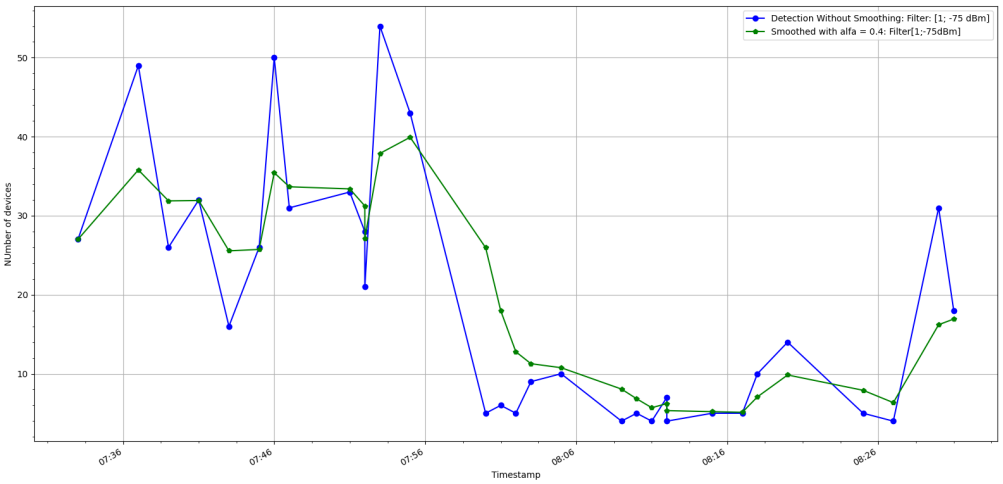

Figure 5.20. Spikes Smoothing $\alpha = 0.4$ - OCTOBER 19 SET

5.4 Confusion Matrix Classification

Our system has high adequacy in classifying the bus utilization into three categories using a confusion matrix. A confusion matrix is a table that is used to describe the performance of a classification model on a set of test data for which the true values are known. It allows the visualization of the performance of our algorithm.

According to GTT [4], the maximum capacity of the bus with ID 33E (BYD Model) is 77 passengers: 21 of them are seated, 55 are standing and 1 with a wheelchair as shown in Figure 5.21.

COVID 19 Regulations asserts that the maximum utilization of public vehicles must be at most 50% of its capacity. The thresholds for the confusion matrix are static according to half of the bus maximum capacity. Hence, we decide to set 3 different classification categories:

- Under-Utilized [Green Zone]: from zero to 15 people on board
- Moderate Utilized [Yellow Zone]: from 16 to 27 on people board
- Highly Utilized [Red Zone]: equal or above 28 people on board

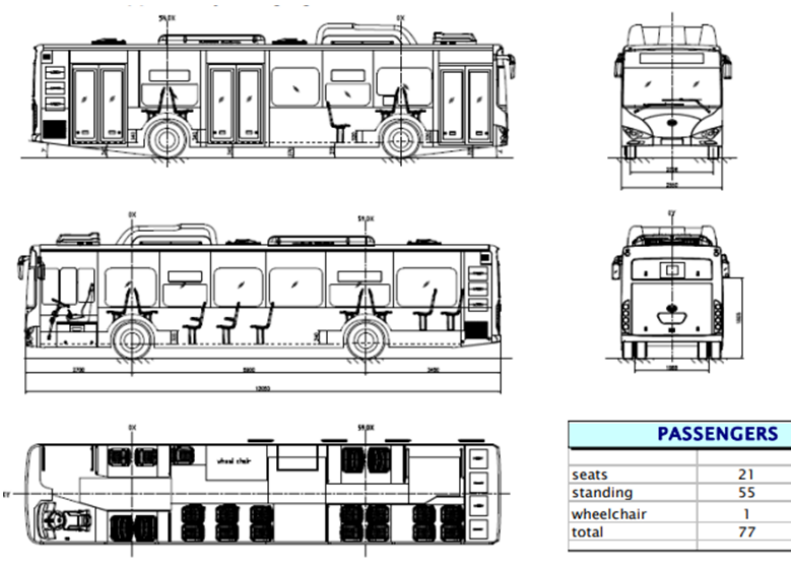

Figure 5.21. Bus layout and boarding capacity (reproduced from [4])

The confusion matrix evaluates the system's overall performance, either the system outcome is overestimating, underestimating or the outcome is in line with the ground truth category class.

The main diagonal of the confusion matrix reflects the accuracy of classifying bus utilization (e.g. underutilized, moderate utilized, or highly utilized). The values under the main diagonal tell the percentage when the system is under-estimating the number of people on board, where the values above the main diagonal show the percentage when the system is over-estimating the number of passengers on board. The weighted average accuracy is calculated by the means of the percentages in the main diagonal, weighted by considering the number of instances of each class. The critical point of view for controlling social distancing is to not under-estimate the number of people on board. Thus we seek for confusion matrix that has low probabilities below the main diagonal.

Since the system can detect only devices with an enabled WiFi interface, a correction factor must be applied to compensate people on board not detected. The correction factor is chosen upon maximizing the weighted average accuracy of the confusion matrix. We add a correction factor ranging from -20% to +20%, where the best correction factor detected is +10% as shown in a sample test in Figure 5.22.

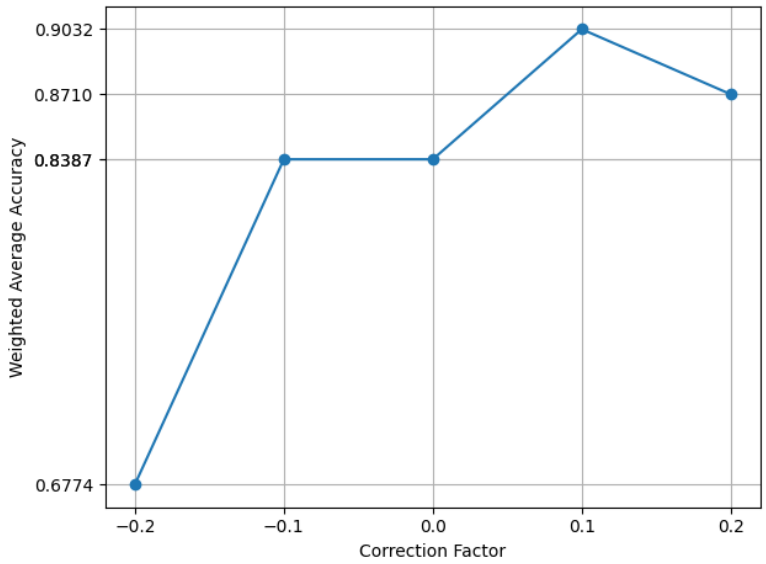

Figure 5.22. Weighted Average Accuracy as function of correction factor

The Tables 5.3, 5.4, 5.5 5.6, 5.7 and 5.8 are showing the confusion matrices for the manual counting sets versus the system detection. Where the corresponding system detection evolution over time versus the manual counting is shown in Figures 5.23, 5.24, 5.25, 5.26, 5.27 and 5.28 respectively.

Our point of interest is to detect when the bus is highly-utilized. As we can see in Table 5.3, the system can classify 100% the highly-utilized category (red-zone). This tells us that when the bus is overutilized, our system can detect that efficiently. The overall weighted average accuracy of the confusion matrix is 84.6%. Weighted average accuracy with a high value tells that the system may have some misestimating but with low probability. However, the percentage of the moderate-utilization class (yellow) classified correctly is 67%, with 33% with under-estimating, but this doesn't reflect the real performance, since the number of instances involved in this category is small compared with other categories as clearly shown in Figure 5.23.

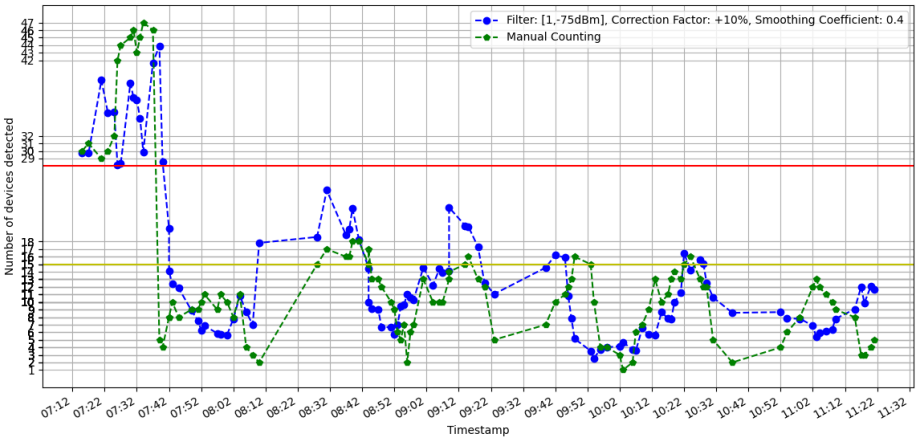

Figure 5.23. OCTOBER 16 - System Detection and Manual Counting evolution over time

		System Detection		
		[0 - 15]	**[16 - 27]**	**>=28**
Manual Counting	**[0 - 15]**	0.84	0.13	0.02
	[16 - 27]	0.33	0.67	0.00
	>=28	0.00	0.00	1.00

Weighted Average Accuracy = 0.846

Table 5.3. OCTOBER 16 - Confusion Matrix

As shown in Table 5.4, the system is 100% classifying the red-zone class. The main drawback here is overestimating the yellow-zone. It is clearly visualized in Figure 5.24 at time instance 08:00 and 08:01. These instances are the first two stops for a new bus cycle. The miss-estimating is since the bus stays for a while at the departing point to restart the cycle again, thus the values of the previous detections will affect the new detection due to the smoothing process. But all in all, the overall performance is not affected significantly, where the weighted average accuracy is relatively high (90.3%), and the system didn't under-estimate the number of people on-board. This tells that the system is efficiently classifying the number of people on board.

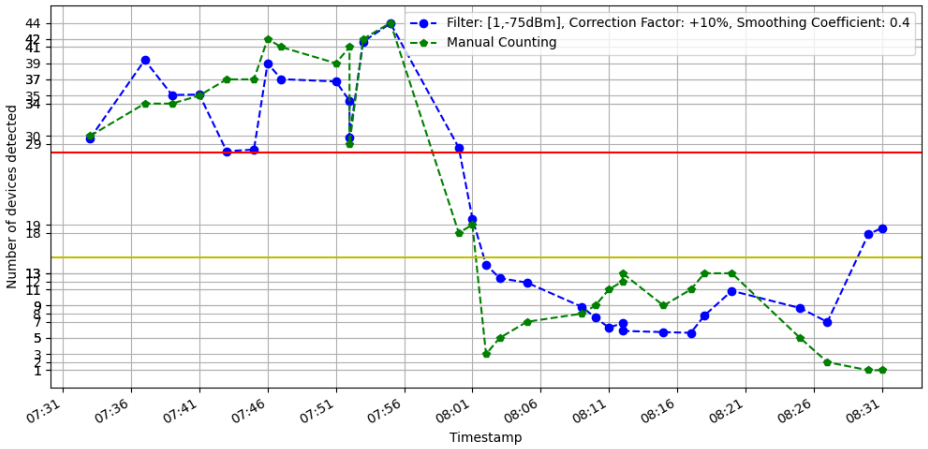

Figure 5.24. OCTOBER 19 - System Outcome versus Manual Counting

		System Detection		
		[0 - 15]	**[16 - 27]**	**>=28**
Manual Counting	**[0 - 15]**	0.88	0.12	0.00
	[16 - 27]	0.00	0.50	0.50
	>=28	0.00	0.00	1.00

Weighted Average Accuracy = 0.903

Table 5.4. OCTOBER 19 - Confusion Matrix

For October 30th manual counting set the weighted average accuracy is 91.9%. Most of the system detections were in the green-zone as the real case. The same scenario repeats for November 2nd , July 26th and July 27th as shown in 5.6, 5.7 and 5.8 respectively. The weighted average accuracy ranges between 83% to 100%, where the interesting point is the system outcome is consistent with the ground truth, where it is neither over-estimating nor under-estimating.

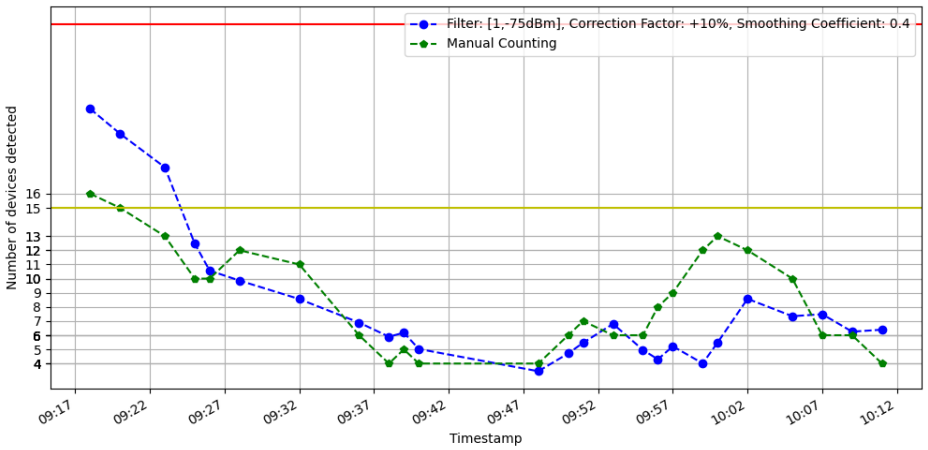

Figure 5.25. OCTOBER 30 - System Outcome versus Manual Counting

		System Detection		
		[0 - 15]	**[16 - 27]**	**>=28**
Manual Counting	**[0 - 15]**	0.92	0.08	0.00
	[16 - 27]	0.00	1.00	0.00
	>=28	0.00	0.00	0.00

Weighted Average Accuracy = 0.919

Table 5.5. OCTOBER 30 - Confusion Matrix

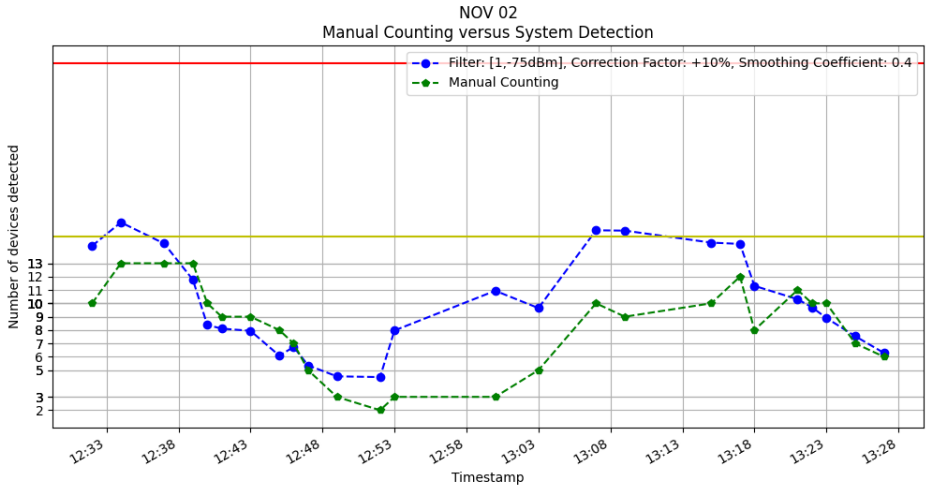

Figure 5.26. NOVEMBER 02 - System Outcome versus Manual Counting

		System Detection		
		[0 - 15]	**[16 - 27]**	**>=28**
Manual Counting	**[0 - 15]**	0.88	0.12	0.00
	[16 - 27]	0.00	0.00	0.00
	>=28	0.00	0.00	0.00

Weighted Average Accuracy = 0.88

Table 5.6. NOVEMBER 02 - Confusion Matrix

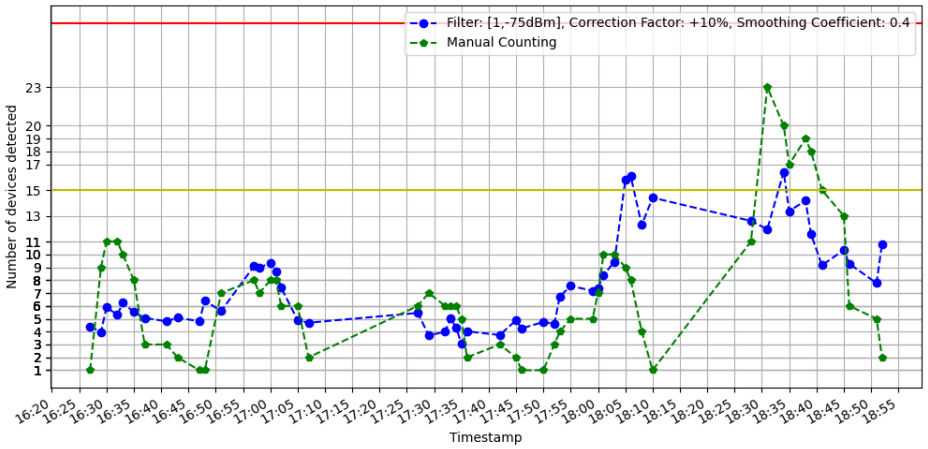

Figure 5.27. JULY 26 - System Outcome versus Manual Counting

		System Detection		
		[0 - 15]	**[16 - 27]**	**>=28**
Manual Counting	**[0 - 15]**	0.96	0.04	0.00
	[16 - 27]	0.80	0.20	0.00
	>=28	0.00	0.00	0.00

Weighted Average Accuracy = 0.884

Table 5.7. JULY 26 - Confusion Matrix

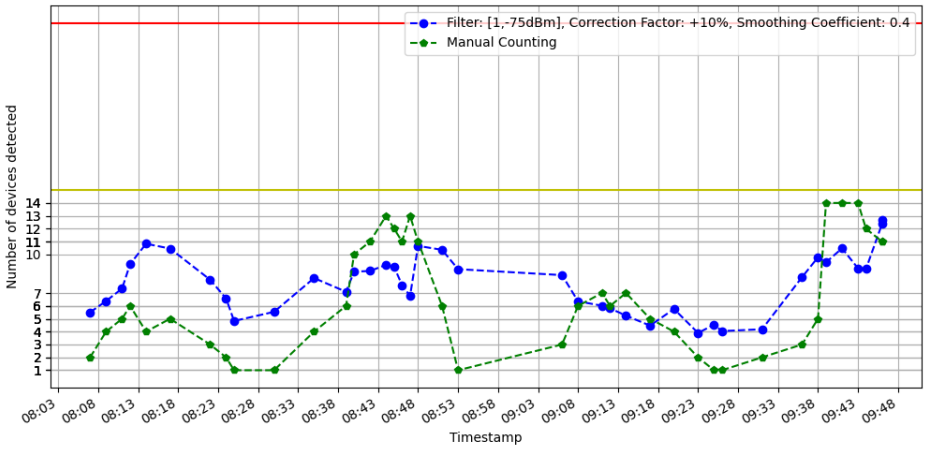

Figure 5.28. JULY 27 - System Outcome versus Manual Counting

		System Detection		
		[0 - 15]	**[16 - 27]**	**>=28**
Manual Counting	**[0 - 15]**	1.00	0.00	0.00
	[16 - 27]	0.00	0.00	0.00
	>=28	0.00	0.00	0.00

Weighted Average Accuracy = 1.00

Table 5.8. JULY 27 - Confusion Matrix

5.5 Frequent Pattern Recognition

As said before, the bus 33E usually run on different lines with different mobility type (e.g. Urban or Sub-Urban). In this section, we try to identify frequent patterns of bus utilization during different days for different lines. Through an hourly heatmap, a frequent pattern can be recognized for each line during either working or vacation days. It is also relevant to see the lockdown regulation effects on these patterns. Blank hour block tells that the bus was not operating in that hour. The darker the color, the higher the number of detections and vice versa. The data collected are from October 15th till 31st of January, however the bus was out of service from November 10th till December 17th.

5.5.1 Urban Mobility Pattern Recognition

As shown in Figure 5.29, the peak bus utilization running at line 6 is at 08:00 and 14:00, in which people are normally going and returning from their work. While in vacation days as shown in 5.30, this peak is no more existing. Over the data captured, the bus is dedicated for line 19 only on working days. The effect of lockdown regulations is clearly shown in Figure 5.31, where the estimated number of people decreased dramatically in the lockdown time. Line 19 is usually utilized by school students since a school is on the path of this line. This proves the peak hours detected are at 07:00 and 14:00 before announcing that lectures are completely online starting October 26th [2].

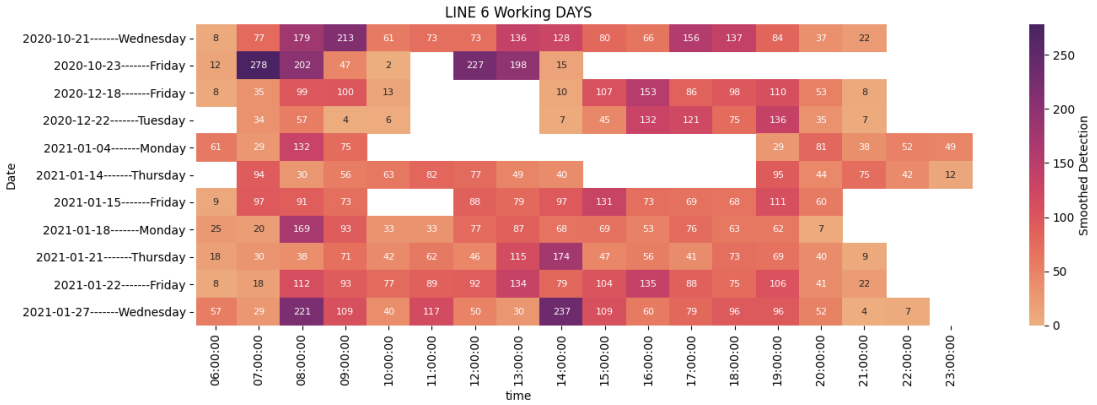

Figure 5.29. Hourly Heatmap of System detection - Line 6 - Working Days

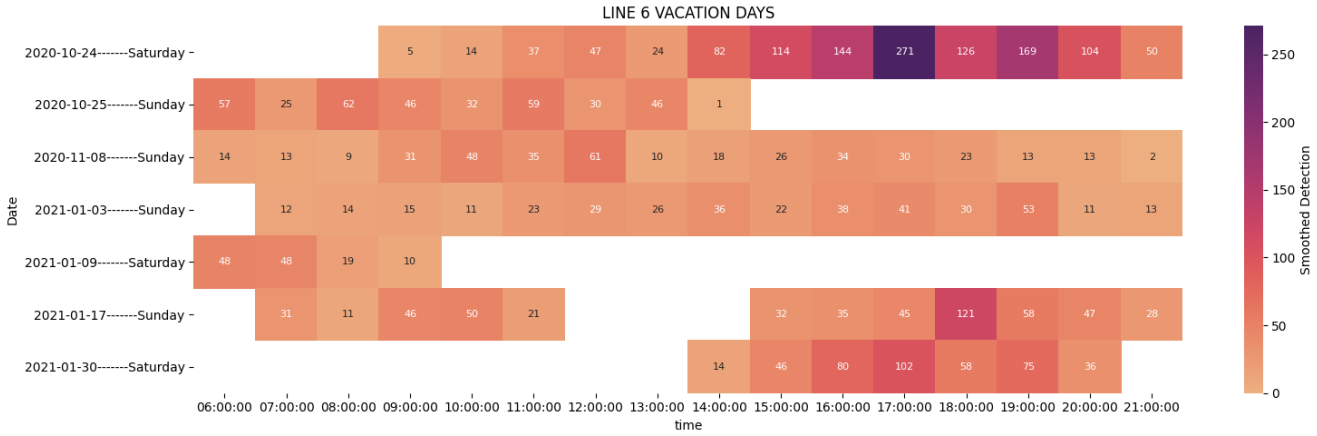

Figure 5.30. Hourly Heatmap of System detection - Line 6 - Vacation Days

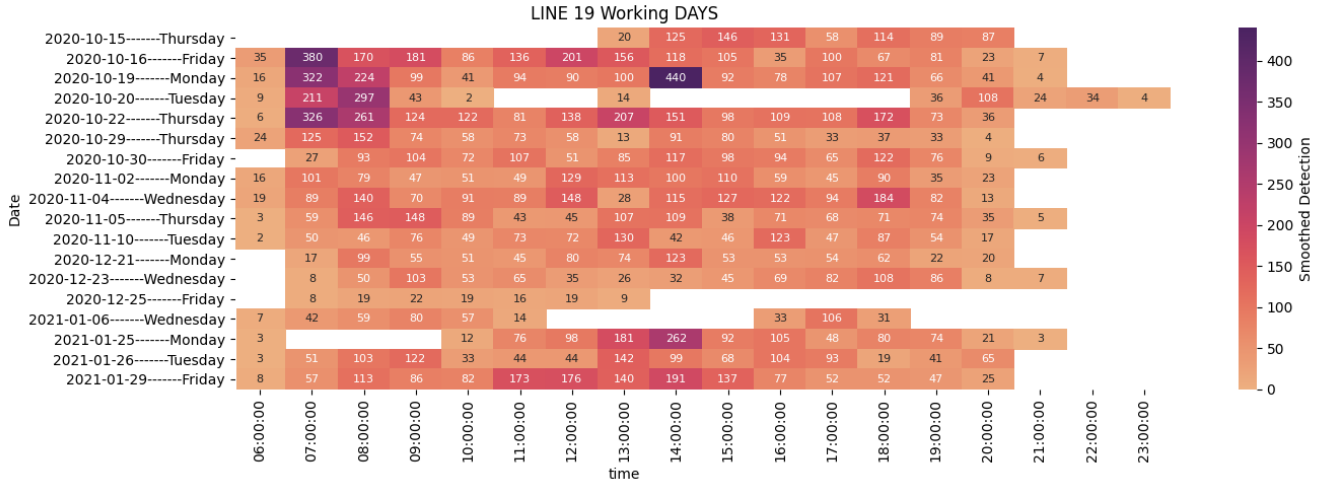

Figure 5.31. Hourly Heatmap of System detection - Line 19 - Working Days

5.5.2 Sub-Urban Mobility Pattern Recognition

The bus operates on SE1, SE2, and 61 lines which are sub-urban paths. These buses are less used by passengers in general, since these buses are going out from the city. Instead, people may use a better mobility option like trains or even their cars to reach the city. The pattern here is challenging to recognize, but it is clearly shown the difference in the utilization between working and vacation days.

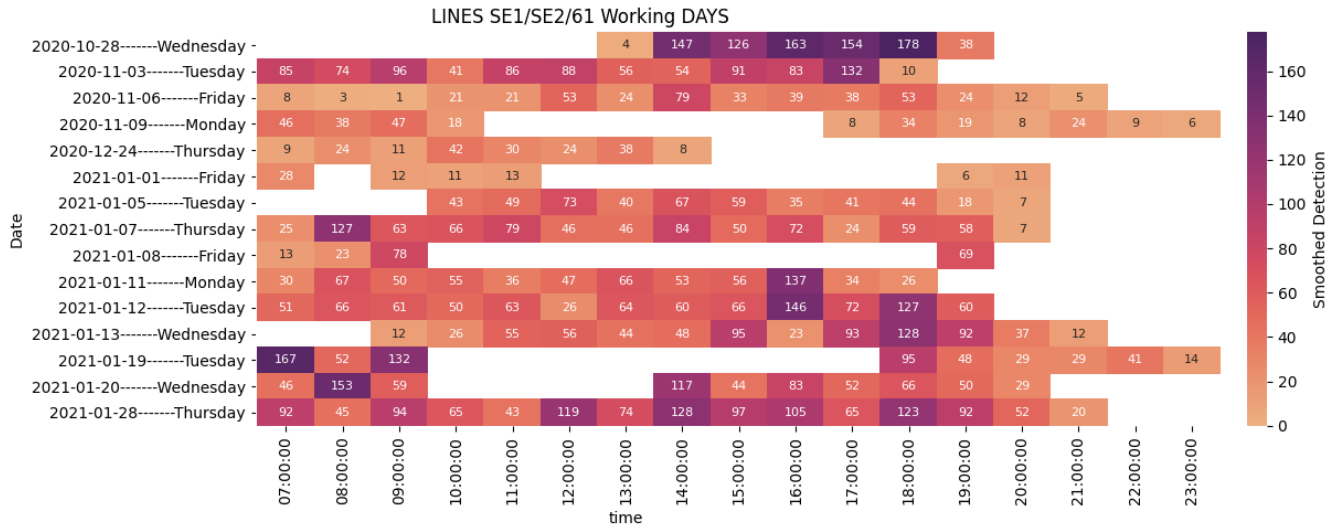

Figure 5.32. Hourly Heatmap of System detection - Suburban Lines - Working Days

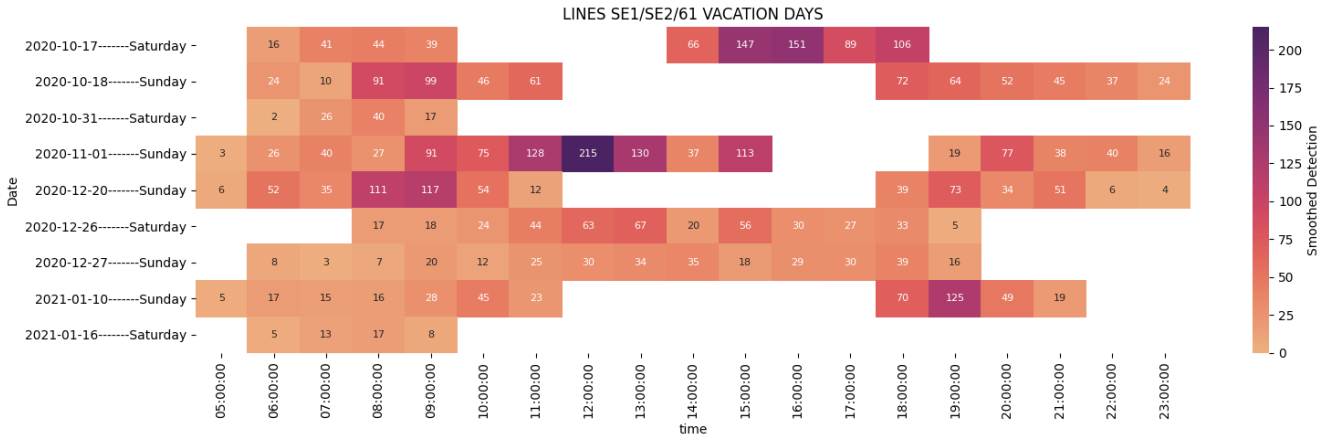

Figure 5.33. Hourly Heatmap of System detection - Suburban Lines - Vacation Days

Chapter 6

Real-Time Dashboard

As said in Section 4.4, the real-time dashboards are implemented using Grafana [6]. This chapter will list in detail all Grafana features and show all the implemented dashboards. Grafana provides a wide variety of feature, as follows:

- The graphs can be customized (e.g. color, shape, etc ..);
- The dashboards are interactive, it is possible to visualize a specific timestamp range and it possible to change the refresh rate;
- Easy configuration of data source, such as MySQL database;
- The dashboards are accessible only after authentication;
- It is possible to generate an alarm if a specific value exceeds a certain threshold.

6.1 Data Visualization in Grafana Dashboard

The following graphs are visualized by Grafana for system outcome from January, 1st, 2021 till January, 31st, 2021. The charts are showing the system outcome with filter combination [minimum MAC occurrence = 1, average received signal power threshold = -75dBm], and applying exponential smoothing with $\alpha = 0.4$, and correction factor = +10%.

Figure 6.1 is showing the main dashboard, which includes 3 charts:

- Line graph of system detection over each time epoch
- Hourly Heatmap shows the average detections over each hour in each day.
- Line graph of the CPU load evolution over time.

As we can see in the line graph of system detection, the peaks are frequent every day in a specific time slot. The CPU load maximum peak doesn't exceed the 45% over all the time interval as shown in the Load consumption line chart. This means that our system is efficiently running by the Raspberry PI 3 without any overloading, whatever the scenario is. More clearly, the peaks are shown in the hourly heatmap also, where each block represent the average of detection for each hour over the day.

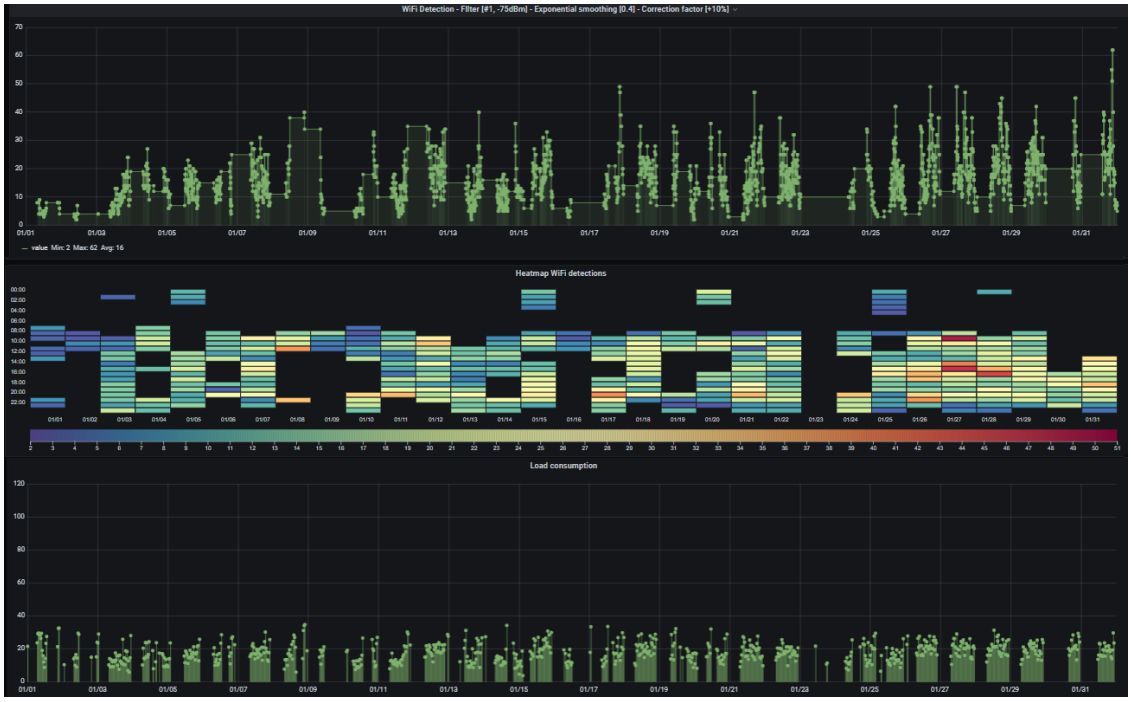

Figure 6.1. Grafana Main Dashboard

All the charts can be zoomed to a specific time interval as shown in Figure 6.2. The red circles here shows the interruption when the bus is not operating.

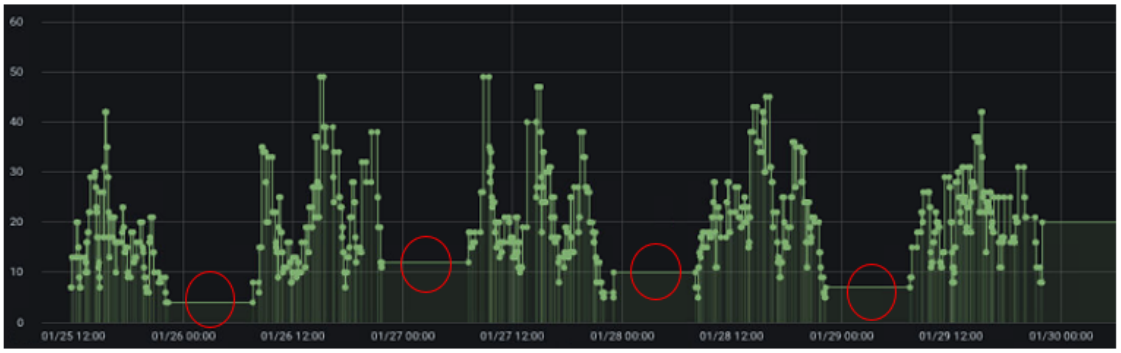

Figure 6.2. More detailed for line chart of system detection

As stated above, we can zoom to any time interval interested in, in Figure 6.3 we zoom for a specific day to see clearly how the system detections are. As expected main trends exist during the day (e.g. at 08:00 and 14:00).

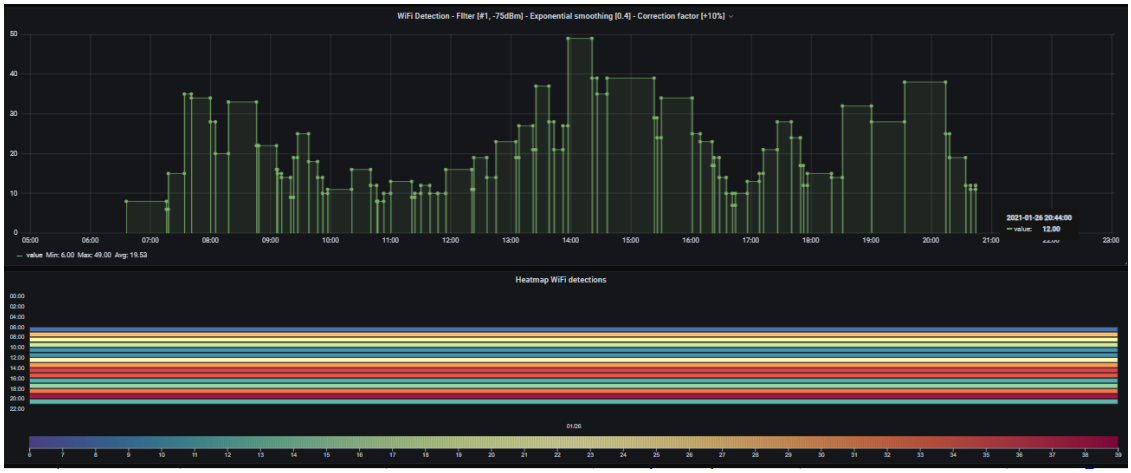

Figure 6.3. 1-Day Zoomed Chart with the hourly heat map

Chapter 7

Conclusion and Future Work

This thesis work proposes an automatic people counting boarding on a public vehicle based on IoT, which infers the number of people on-board based on WiFi probe requests received by a sniffer installed on the bus. Mobile devices nowadays are applying MAC randomization, thus capturing a unique MAC as a single person not true anymore. For this reason, we build a de-randomization algorithm to understand which MAC addresses captured are most likely belong to the same device. This algorithm is recursive that returns a score based on timestamps of the frames, the sequence number of each frame, and the received signal power for each one.

The results are entirely good, principally in terms of bus utilization classification. The interesting point is to detect the high utilization of people boarding to control the social distancing. The system can classify 100% when the bus is highly-utilized. Whereas the derandomization algorithm was magically improving the order of detection accuracy. However, without applying the algorithm, the estimation of the number of people on board was in the order of hundreds, which is relatively not reasonable. Some events may affect the system performance, thus we implement an average smoothing process that depends on the previous detections to limit the effect of such events. Furthermore, a correction factor is applied to compensate the people on board having no smart device, or turning off the WiFi interface.

For sure, this prototype done can also be a starting point for other projects, such as estimating in real-time the number of people in other environments, such as airports, hospitals, schools, train stations, etc... It would be very useful to analyze the number of people present in a given area, to allocate appropriate resources in each time slot. For any public transportation company, they can re-plan the timetables based on frequent peaks observed recently. This would improve the company's overall performance.

Bibliography

[1] Android 6.0 changes. https://developer.android.com/about/versions/marshmallow/android-6.0-changes.

[2] Covid-19 october 26th italian authorities announcement. https://www.garda.com/crisis24/news-alerts/392941/italy-authorities-announce-new-covid-19-restrictions-nationwide-from-october-2

[3] Do raspberry pi 3 supports wifi monitor mode? https://www.raspberrypi.org/forums/viewtopic.php?t=222151.

[4] Electric transport in torino gtt experience. https://ec.europa.eu/environment/gpp/pdf/Zanini.pdf.

[5] Google chart website. https://developers.google.com/chart.

[6] Grafana website. https://grafana.com/.

[7] Probe request frame. https://www.oreilly.com/library/view/80211-wireless-networks/0596100523/ch04.html.

[8] Socket - low-level networking interface. https://docs.python.org/3/library/socket.html#socket.AF_INET.

[9] Philip J. Basford, Steven J. Johnston, Colin S. Perkins, Tony Garnock-Jones, Fung Po Tso, Dimitrios Pezaros, Robert D. Mullins, Eiko Yoneki, Jeremy Singer, and Simon J. Cox. Performance analysis of single board computer clusters. *Future Generation Computer Systems*, 102:278–291, 2020.

[10] Julien Freudiger. How talkative is your mobile device? an experimental study of wi-fi probe requests. New York, NY, USA, 2015. Association for Computing Machinery.

[11] Xiaolin Gu, Wenjia Wu, Xiaodan Gu, Zhen Ling, Ming Yang, and Aibo Song. Probe request based device identification attack and defense. *Sensors*, 20(16), 2020.

[12] Hande Hong, Girisha Durrel De Silva, and Mun Choon Chan. Crowdprobe: Non-invasive crowd monitoring with wi-fi probe. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(3), September 2018.

[13] Jeremy Martin, Travis Mayberry, Collin Donahue, Lucas Foppe, Lamont Brown, Chadwick Riggins, Erik C. Rye, and Dane Brown. A study of mac address randomization in mobile devices and when it fails. *Proceedings on Privacy Enhancing Technologies*, 2017(4):365 – 383, 01 Oct. 2017.

[14] Mathieu Cunche Leonardo Cardoso Frank Piessens Mathy Vanhoef, Célestin Matte. Why mac address randomization is not enough: An analysis of wi-fi network discovery mechanisms. May 2016.

[15] Michele Nitti, Francesca Pinna, Lucia Pintor, Virginia Pilloni, and Benedetto Barabino. iabacus: A wi-fi-based automatic bus passenger counting system. *Energies*, 13(6), 2020.

[16] L. Oliveira, D. Schneider, J. De Souza, and W. Shen. Mobile device detection through wifi probe request analysis. *IEEE Access*, 7:98579–98588, 2019.