Politecnico di Torino

POLITECNICO DI TORINO

Master Degree course in Computer Engineering

Master Degree Thesis

Enhancing Crowd-Monitoring Through WiFi Fingerprint Analysis

Supervisors
Prof. Claudio Ettore Casetti
Prof. Paolo Giaccone
PhD. Riccardo Rusca

Candidato
Diego Gasco

Academic Year 2022-2023

Acknowledgements

I would like to thank my supervisors, Claudio Ettore Casetti and Paolo Giaccone, and my mentor Riccardo Rusca for their availability, enthusiasm, and professionalism, they put into each part of this work. Also, a special thanks go to my parents, my girlfriend, and all my friends, who have supported and encouraged me all these years.

Abstract

The proliferation of smartphones, IoT devices, and other modern technologies has transformed cities into interconnected ecosystems, generating vast amounts of data. Accurately estimating crowds and counting people has become crucial for urban planners, transportation managers, and security agencies. By leveraging real-time data from various sources, decision-makers can optimize resource allocation, enhance security measures, improve customer experiences, and create more efficient urban environments. Capturing and analyzing network traffic has emerged as a valuable method for accurately estimating people's presence in specific areas. WiFi and Bluetooth are the two main types of signals that can be inspected, with WiFi being the preferred option for privacy reasons. WiFi Probe Requests, emitted by devices when they search for available WiFi networks, provide valuable data on the number and movement of people in specific areas. The main focus of this thesis is on estimating people counts through capturing, processing, and analyzing these kinds of messages. Firstly, a synthetic Probe Requests generator was developed to replicate Probe Requests sent by a customizable number of different devices. The generator is designed to simulate realistic Probe Request traces, based on data from real case scenarios. By leveraging this simulator, it is possible to provide a precise ground truth reference for the number of devices present in a given area. This approach enhances both the evaluation phase of counting methods and the training phase for machine learning techniques. Secondly, crowd-monitoring techniques have been employed to address the challenge of people counting. Since the probe requests' MAC address is randomized by modern operating systems, counting based solely on different addresses is not feasible. Instead, the focus has shifted to handling complex data patterns and extracting meaningful insights from messages. Based on an in-depth understanding of probe request fields and time features, two advanced frameworks have been successfully developed. The parameters of the algorithms have undergone rigorous training using vast amounts of data generated by the Probe Requests generator, ensuring optimal performance and accuracy. Drawing inspiration from clustering methodologies, our systems adopt a similar approach to analyze probe requests. By leveraging the power of these techniques, the frameworks can categorize and group probe requests based on their source device or vendor model. For one of the two implemented systems, the messages' time features have been taken into account to estimate the number of devices present. The developed methods provide a good approximation for estimating the crowd in a given area, demonstrating that these approaches closely align the retrieved number of people with the ground truth provided by the generator. The outcomes of this research have practical applications across various domains. In retail analytics, accurate people counting and flow estimation provide insights into customer behavior,

optimizing store layouts, staffing, and marketing strategies. Another important aspect is to enhance safety in public environments such as events, pedestrian traffic in urban areas, and emergencies. Additionally, the development of a synthetic Probe Requests generator contributes to the advancement of crowd-estimation techniques, providing a valuable tool for evaluating and improving counting methods based on WiFi probe requests.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, the proliferation of Internet-of-Things (IoT) and smart devices, offers the possibility to track and estimate the number of people in a certain area. For the activities and situations that can be regulated by the knowledge of the number of individuals, crowd-monitoring is a key aspect. When considering the largest cities globally, it becomes evident that they are home to a substantial number of smart devices, primarily due to their high population density and industrial activity. In recent years, the proliferation of smart and IoT devices has become a necessity for individuals, not just for professional reasons but also for personal interests and everyday convenience. As a result, the emergence of these interconnected urban environments, commonly referred to as smart cities, has become increasingly prevalent. The result of this transformation leads to population movements through various information systems and infrastructures. In this sense, people flow estimation could become very useful for a better quality of life in cities. Examples of real-world situations include improving safety in public settings like events, pedestrian traffic in cities, and emergencies, as well as optimizing energy usage in enclosed spaces. Accurate crowd monitoring can also prove beneficial for other purposes, such as analyzing customer behavior to inform marketing strategies. Understanding and managing crowd dynamics within smart cities can be the right way to exploit the potential of these new urban environments.

Before 2014-2015, tracking mobile devices was a simple task due to the little importance given to the protection and privacy of user data. At that time the estimation of the number of people present in a certain urban area, such as train stations, public gardens, streets, and offices, could be done simply by collecting network messages that smart devices send periodically. There was some information inside the captured traffic that permitted the identification of the source devices inside a certain area. Very few masking techniques were used with the result that these messages might be collected in groups, one for each device. Fortunately, in recent years, companies and organizations have been becoming more aware of this

topic and a lot of measures have been developed for the protection of users' privacy. Within the European Union Area, a significant and pivotal stride towards bolstering data protection and safeguarding individual privacy materialized in the year 2018, marked by the establishment of the General Data Protection Regulation (commonly referred to as GDPR [8]). This legislative enactment has given individuals greater control over their personal information and imposed strict ethical and responsible data management obligations on organizations. This regulatory framework is meticulously crafted rules governing the spectrum of data-related activities, from data collection, through utilization, and finally sharing. All the regulation is surrounded by a rigorous system of penalties designed to deter and penalize any infractions or non-compliance.

The purpose of this thesis work is to find a solution for two problems related to crowd-monitoring: the difficulty of collecting fingerprint traces and a solid method to estimate the presence of people in a specific area. The first step was the construction of a precise and easy-to-use system, able to create synthetic data for the emulation of the real probe request messages that devices periodically send. This information could be used to train and test counting algorithms for crowd-tracking. The effort required to collect real data is not indifferent. It involves the usage of sensors that need to be programmed and installed with the risk of bumping into poor crowded situations, where data collected can not be many. In addition, another issue encountered in the traditional collecting methods is the difficulty of knowing the precise number of devices that produce the messages, in particular for situations where the number of people is very high. To solve these problems, a WiFi traces generator was devised, representing a valuable improvement as it offers realistic message traces without the need to collect them in a real-world environment. The generator also resolves the issue of determining the precise number of devices present, as it produces traces with user-controlled settings for the active number of devices. The second goal of this thesis work is a solution for actual crowd-monitoring through the knowledge acquired from the message composition. The older methods based on collecting data and dividing messages through sensitive information are not possible anymore, new methods must be investigated and, most importantly, they must be privacy conscious as well. Thanks to the traces provided by the generator, we can emulate different types of devices, whose behavior is accurately modeled. The measurement of the counting frameworks' effectiveness becomes simple since the synthetic data provides the precise number of devices present in the environment. In our specific case, the development of the generator required deep knowledge of the message's fields and time features. These two domains offer the potential for the development of robust systems capable of categorizing messages into groups and implementing counting techniques based on rigorous mathematical formulations. This was the second main contribution of this thesis, the development of two frameworks for people counting. There

exist other crowd-tracking methods, such as computer vision systems, LiDAR and infrared sensors, and security cameras. All of them pose challenges related to hardware equipment, since the data collection and processing may require expensive components. Another crucial aspect is the sensitivity of these methods in individual privacy reservations. All of these issues make the proposed WiFi messages methodologies an attractive alternative for this research field.

Chapter 2

Crowd-monitoring

This chapter will provide an overview of the state of the art for people tracking methods, describing different systems and the theory behind them. All the techniques covered will be analyzed for the effort and costs in the development phase, and compliance with the privacy requirements. The technical explanation will be coupled with a detailed insight into the strengths and weaknesses of each system. A deep focus will be on WiFi probe request messages 2.4, broadly used for this thesis work. The chapter is structured as follows: Section 2.1 covers the video camera surveillance systems, Section 2.2 describes the modern systems that exploit computer vision technologies, Section 2.3 and 2.3.1 offer an overview about recent systems based on infrared and LiDAR scanners, and lastly, Sections 2.4 and 2.5 provides the description of how WiFi probe requests can be used for these purposes.

2.1 Video camera

Video camera systems are widely used for crowd-monitoring and surveillance purposes, they consist of multiple cameras strategically positioned in areas where crowd activity is expected or needs to be monitored. The main goal is to capture and record visual information from specific locations or areas. Video cameras are strategically placed to cover a wide area or specific locations where crowd-monitoring is required. They can be mounted on buildings, poles, or other structures to achieve optimal coverage as can be seen in Figure 2.1

The strength of these systems is the continuous capture of images and videos about crowd movements, behavior, and interactions in real time. This allows operators or automated systems to monitor and analyze people's movements and dynamics. One important issue is the generation of vast amounts of visual data, which has to be stored and retrieved for analysis.

An example of the usage of video camera surveillance systems for people tracking is described in [5]. A human partial body detector is used to detect upper body images in the overlapping area captured by two cameras. Next, a partial identification network is involved to classify these images and record the number of upper-body image pairs with a high matching degree. This process involves complex algorithms to compute and extract low-level features of the video frames. Lastly, the total number of people in the captured sequence is calculated as the sum, taking into account the presence of duplicates.

Despite the good results obtained, this method presents some problems, such as the need for complex hardware to be managed, and, more importantly, the manipulation of human-sensitive data, like face or body pictures. The difficulty of acting in outdoor places might become problematic for a good generalization of

Figure 2.1. Example of a multi-camera surveillance system.

this solution. In addition, is important to consider that this system can be very expensive due to the particular hardware used.

2.2 Computer vision

Computer vision is a multi-disciplinary field that aims to give computers a high-level understanding of multimedia resources, such as images and videos. It involves systems for collecting data, engineering processes to transform information, algorithms to extract main features, and mathematical procedures to reach a comprehension of the resource content. Computer vision methodologies include scene reconstruction, video tracking, object recognition, motion estimation, and object detection. It became relevant in the last decade due to the diffusion of Machine Learning (in particular the sub-field of Deep Learning) usage that permits the treatment of this kind of data more efficiently and in a more standard and automatic way. Machine learning is a field of Artificial Intelligence (AI), that strictly connects computer science and statistics. The focus is on developing algorithms that enable a computer to learn from data and make predictions or decisions without human supervision. It involves the creation of mathematical models that analyze and extract patterns from large collections of data (called datasets) to identify relationships and make predictions. Typically, machine learning requires a training phase as a means to find the optimal model parameters for the task. Deep Learning is a branch of machine learning that exploits the potential of particular structures composed of interconnected nodes that mimic the behavior of human neurons, called neural networks. In recent years, deep learning's popularity has increased a lot due to the growth of computing power, which allows the collection and processing of huge datasets. Computer vision systems are often built on top of deep learning models so that they can recognize particular patterns in luminance and colors of the images provided. Figure 2.2 shows how deep learning can be applied to images. Some

Figure 2.2. Computer vision with Deep Learning technique.

works employ computer vision for crowd-monitoring, starting from data collected with the support of a multi-camera system. In [19], the developed method focuses on overhead view-based detection through two different types of feature extraction. Data processing is one of the more difficult steps in the system because images need a background subtraction to extract the foreground object. The research of the optimal parameters of the mathematical model could be very difficult and may require a very long and difficult training process. At the end an algorithm is employed to classify image contents in a binary fashion: as a person or non-person.

Nowadays, computer vision is a very active research field, and the frameworks developed are achieving good results. In the context of people tracking systems, unfortunately, these processes involve the collection of people's images that surely contain sensitive information. Furthermore, the need for complex algorithms, such as neural networks and classifiers, forces the usage of high-performance hardware. Since GDPR ([8]) aims to preserve people's privacy inside the collected data, these kinds of solutions may not be appropriate. Images collected contain human details such as clothes, objects, and, more importantly, faces. For this reason, computer vision could be not privacy-compliant.

2.3 Infrared sensors

Infrared sensors are a class of devices that detect and measure infrared radiation emitted or reflected by nearby objects. Infrared radiation is a form of electromagnetic radiation with longer wavelengths w.r.t. those of visible light but shorter w.r.t. microwaves. We can find this technology in various fields, including industrial, military, medical, consumer electronics, and environmental monitoring. The

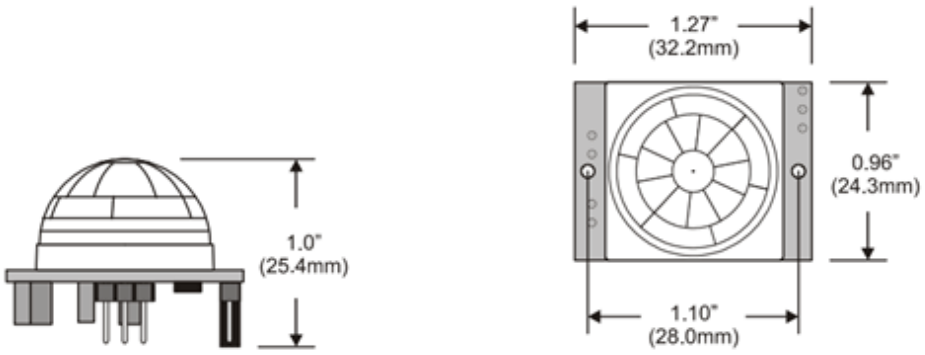

Figure 2.3. An example of Passive Infrared (PIR) sensor (reproduced from [24]).

energy carried by infrared radiation causes molecules in materials to vibrate, leading to temperature changes. These temperature variations can be harnessed by special sensors to detect and measure the radiation. There are two main types of infrared sensors:

- Thermal Infrared Sensors: also known as Passive Infrared (PIR) sensors, are based on the principle that all objects with a temperature above absolute zero emit infrared radiation. These sensors typically consist of an infrared detector element that converts the incoming radiation into an electrical signal. In Figure 2.3 there is an example of a passive infrared sensor.

- Active Infrared Sensors: they use an infrared light source to emit radiation, which is then reflected to the sensor. This latter measures the time taken for the emitted radiation to return, allowing the calculation of the distance to the object. One of the main uses of this type of sensor is in the LiDAR systems.

There are many cases of applications that exploit the use of thermal infrared sensors for people tracking systems. One example is described in [6] where the proposed system combines a PIR sensor and a video camera. PIR sensors provide additional information about human motion in the area w.r.t. multi-camera surveillance systems [5,19] so that the flow of people can be studied and analyzed. Thanks to the collected data about temperature changes, they can classify different types of human motion, specifically entry/exit motions and ordinary activities. Camera-based face detection is taken into account to combine it with the aforementioned system. This approach reduces false negatives and false positives generated by the camera-only systems and avoids heavy image processing.

Another example of crowd-monitoring through PIR sensors was done in [22]. The device used is an infrared sensor to detect people's direction and occupancy in various spaces of a building while respecting privacy. The sensor captures thermal images and temperature gradients, and the data is processed using a particular pattern recognition algorithm. This latter generates output based on the detected behaviors, corresponding to events such as entry, exit, or hold.

In both presented works, there are some limitations associated with these technologies. Firstly, the expensive hardware involved in the systems, since infrared sensors may have a high cost and the developed algorithms for processing the information require high-performance systems to be applied. Then, as for other solutions, in [6] and more in general in multi-camera systems, the collection and manipulation of highly sensitive data is a problem because these systems risk to be not compliant with GDPR. The images and videos captured probably contain some people's personal information like objects owned and faces.

2.3.1 LiDAR scanners

LiDAR (Light Detection and Ranging) scanners are remote sensing devices that use laser light to measure distances and create high-resolution 3D representations of the surrounding environment. This technology has reached significant popularity in various fields due to its precise measurements and ability to capture detailed spatial data. The basic principle employed in time-of-flight measurement is as follows: the device emits laser pulses toward the target area, and the time taken for the light to bounce back to the sensor is recorded. In this way, the distance between the sensor and the object can be accurately calculated. The output of these systems is a 3D point cloud that represents the surfaces and objects in the scanned area. In Figure 2.4 there is an example of the structure of a LiDAR scanner.

Recent works [3, 11] employed LiDAR sensors for crowd-monitoring. In particular, in [11] the authors discuss a system for detecting people in video streams. The system aims to address privacy concerns with traditional camera-based methods and would like to overtake the limitations brought by illumination variances. The proposed system uses a LiDAR sensor to capture coarse human shapes by concatenating multiple two-dimensional range scans. All the framework is designed to work under dynamically illuminated conditions. One of the most important innovations w.r.t. old infrared sensor systems, is that this technique provides enough data for object classification without revealing individual identities. The implementation involves data preprocessing and an object classification algorithm that is used for two binary classification tasks: first distinguishing humans from non-humans and then determining the walking direction of humans.

Talking about [3], the system presented is designed for corridors. A single LiDAR

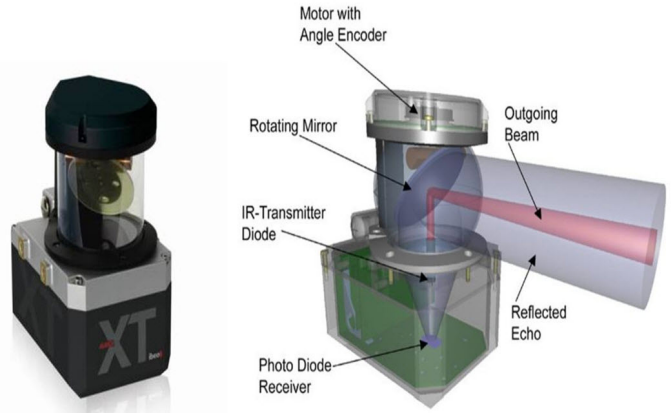

Figure 2.4. An example of LiDAR sensor (reproduced from [14]).

scanner is coupled with a mirror reflection device, enabling the counting of passing pedestrians and the detection of their direction. Subsequently, a technique is used to identify individuals.

The main contribution of LiDAR scanners is privacy management: in this case, there is no collection of sensitive data. The main problems are, once again, the environmental use cases that remain too restrictive, the hardware cost, and management.

2.4 Probe requests

One of the main characteristics of smart devices is the possibility to establish a connection with other devices or with the internet network through an Access Point (AP). In this context, the specific type of messages exchanged inside the network are called Probe Requests (PRs) and, depending on the type of communication, they can be WiFi or Bluetooth.

2.4.1 WiFi

When a WiFi-enabled device is not presently linked to a network, there are two primary methods through which mobile devices can uncover available networks:

- Passive scanning: involves devices constantly listening for beacons sent by APs to identify known networks.
- Active scanning: involves devices actively searching for APs by transmitting a particular type of management frames, called probe requests, on 802.11b/g/n channels.

Passive scanning is not particularly efficient, so proactive scanning has become the preferred approach nowadays. With active scanning a device periodically sends out probe request frames to scan the surrounding environment for available APs. These messages are typically sent in groups composed of one, two, three, or four packets, called bursts. Typically, messages are transmitted across a total of 13 channels, compliant with the 802.11 standard. These channels enable nearby access

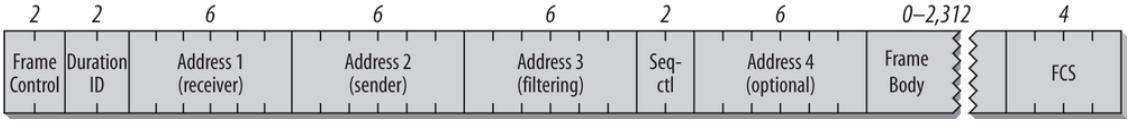

Figure 2.5. Structure of probe request frame (reproduced from [25]).

points to both receive and reply using probe response frames. By analyzing the probe responses, the device can assess the available networks and determine the best access point to connect to, based on multiple factors like signal quality and security. The client will not receive any response in one of these possible cases: there are no APs in proximity or the APs are in passive mode. In Figure 2.5 there is an example of a probe requests frame.

Probe Requests contain certain data that may reveal users' personal information, thereby violating their privacy. Examples of these fields include the MAC address, sequence number, and the list of SSIDs. In the following sections, we will delve into the details of each of these fields and the privacy concerns they pose.

2.4.2 Bluetooth

Bluetooth devices operate in the globally unlicensed frequency range of 2.4 GHz, enabling short-range wireless communication. When a Bluetooth-enabled device has its Bluetooth interface activated, it can be in one of two states:

- Discoverable/visible.
- NOT discoverable/NOT visible.

If a target device is not in *visible* mode, the only way to initiate the pairing process is by knowing its MAC address which serves as a unique identifier for each Bluetooth device. In a very similar way to WiFi probe requests, the Bluetooth protocol utilizes a specific mode called *inquiry* to discover other devices: a device sends out inquiry packets, and other devices within range and in inquiry scan mode can respond with inquiry replies. In this way, the devices have information that gathers the necessary details to establish a connection. According to the Bluetooth specification [1], the inquiry procedure has a maximum duration of 10.24 seconds, during which the inquiring device should be capable of detecting all devices within its range.

Talking about privacy issues, Bluetooth protocol presents an important problem due to the sensitive information exchanged during the *inquiry* mode. By watching the discovering phase it can be seen that the custom Bluetooth device names (e.g., "Device of ...") are visible to the other participants of the communication. This is the main issue in collecting and processing this particular network data. Special methodologies should be applied to keep the cited information to be GDPR-compliant [8].

2.5 WiFi probe requests

2.5.1 Overview

WiFi probe request headers are composed of different fields, according to the 802.11 standard. Each one of these fields has a specific role in the communication between the client and the AP, they all bring information about settings and characteristics that the two parties must take into account to establish a working exchange of messages. Here is a list of the main fields in probe request frames:

- *MAC (Media Access Control) Address*: this address refers to a unique identifier assigned to a network interface controller (NIC). It is a six-byte address that distinguishes a device on the network. MAC addresses are assigned by the vendor which must ensure that they are globally unique. This address is mainly used for communication within a local area network (LAN).

- *Sequence Number*: a unique identifier assigned to each packet sent over a network connection. It is a value attached to the packet header that helps in the proper ordering and reconstruction of the packets at the receiving end. Probe request messages have sequence numbers of 12 bits, so the possible values go from 0 to 4096.

- *SSID (Service Set Identifier)*: a string of characters that serves as the name of a wireless network. It is used to identify and differentiate wireless networks in an area. When devices scan for available networks, they can detect and display the SSIDs of nearby networks. Clients announce the list of knowing SSIDs to try to connect with a specific wireless network.

- *VHT (Very High Throughput) Capabilities*: refer to the capabilities of a WiFi device that supports the IEEE 802.11ac standard, also known as Wi-Fi 5. This particular field in a probe request brings information about the device's support for various features, such as supported channel widths, spatial streams, and modulation schemes. These capabilities enable higher data rates and improved throughput in wireless communications. The throughput in particular refers to the quantity of data that can be effectively conveyed across a network within a specific timeframe.

- *HT (High Throughput) Capabilities*: pertain to the capabilities of a WiFi device that supports the IEEE 802.11n standard, also known as Wi-Fi 4. The information brought by this particular field, includes channel bonding, MIMO (Multiple-Input Multiple-Output) configurations, and supported data rates. These capabilities enhance the data transmission speed and overall performance of the wireless network.

- *Extended Capabilities*: in a WiFi probe request refer to additional features and capabilities supported by the WiFi device. These data go beyond the basic capabilities specified in the VHT and HT fields. The extended capabilities provide information about advanced features such as beamforming, spatial reuse, or other proprietary extensions implemented by specific vendors. It allows devices to communicate their enhanced functionalities to other devices in the network.

- *WPS (Wi-Fi Protected Setup)*: a network security standard that simplifies the process of connecting devices to a Wi-Fi network. It allows users to easily establish a secure connection by using a PIN, a push-button, or NFC (Near Field Communication). The WPS field in a probe request message indicates whether the device supports Wi-Fi Protected Setup and provides information about the supported methods for establishing a connection.

- *UUID-E (Universally Unique Identifier-Extended)*: is used to differentiate between different devices. Within this field lies an exclusive identifier allocated to the device, enabling the recognition and differentiation of this particular device from others by fellow devices or network systems. The UUID-E may be generated by the device or assigned by the network administrator.

2.5.2 People fingerprint and privacy

Studies from both [30] and [16] underline that the management of MAC addresses is an important security point for all device vendors. The MAC address is a unique identifier assigned to a device's network interface controller (NIC). Figure 2.6 reports the basic structure of MAC address, it is composed of 6 octets (i.e., 6 Bytes) that are divided into two halves:

- The most significant half refers to the Organisationally Unique Identifier (OUI), a globally unique identifier that IEEE assigns to a vendor.

- The least significant half refers to NIC-specific fields.

The second least significant bit of a MAC address is the Globally/Locally bit identifier which determines whether the MAC address is globally unique or locally administered. If this bit is set to 0, it indicates that the MAC address is globally unique. These addresses are assigned by the Institute of Electrical and Electronics Engineers (IEEE) and are unique worldwide. Otherwise, if the bit is set to 1, it indicates that the MAC address is locally administered. In this case, the address can be assigned by network administrators to devices within a local network, but they are not guaranteed to be globally unique. In the probe request use case, the MAC address can have both the Globally/Locally bit identifier configuration. Since the probe requests are sent periodically by a device to discover available access points,

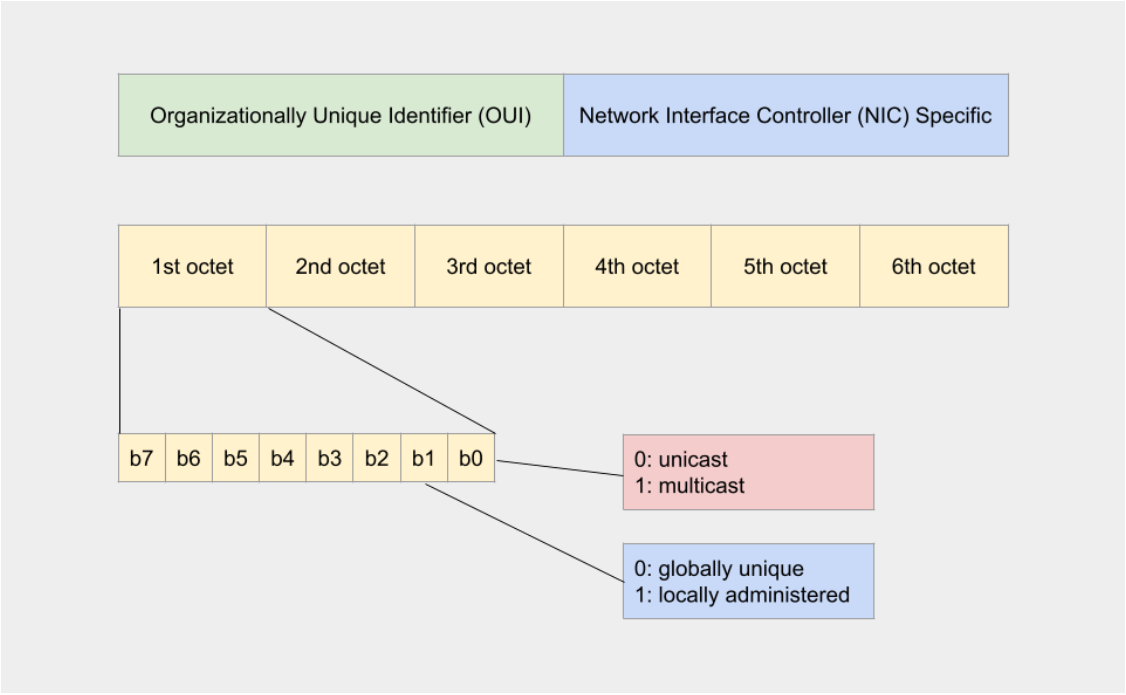

Figure 2.6. Reproduction of the MAC address structure.

the MAC address becomes a possible way for hackers to perform attacks inside the wireless communication. For this reason, the GDPR [8] considers this identifier as sensitive data for user privacy. Nowadays, almost all operating systems adopt a particular procedure to avoid possible malicious actions: MAC address randomization. Vendors started randomizing the second least significant half and keeping the OUI's part unmodified. In recent years the randomization process is increasingly been applied to the whole address, except for Globally/Locally and Unicast/Multicast bit identifiers. In [30], it was observed that each burst produced by a device, uses a different MAC address w.r.t. the previous and next ones. By applying randomization, user privacy reaches a higher level of safety.

Another interesting aside that can be done is the management of sequence numbers. This element of packet headers is also exposed to a series of possible network attacks since it can become an important fingerprint element. As for the MAC address, the operating systems started to adopt a particular behavior, clearly explained in [16]: inside each burst, the sequence numbers are growing from one packet to the next, but the starting one is randomly chosen (i.e., it has no relation with the ones used in previous bursts or that will be used in next bursts).

The observations and experiments done in [30] demonstrate also that the administration of the SSIDs is crucial for a good privacy-conscious system. The clients use the SSID fields in probe requests to connect to a specific wireless network. In particular, devices may search for a known AP with which they connected in the past, so they could share their list of SSIDs. This information might be used to retrieve some information about private places visited by the user. For this reason, nowadays the use of SSIDs inside the probe request frames is not so widespread in the APs searching phase.

The WPS and UUID-E management is a key point for what regards sensitive data. The UUID-E is a unique identifier that enables other devices or network systems to identify and differentiate it from other devices. It is important to remember that the UUID-E field is present only in case WPS is set within the probe request. Even in this situation, we are in the presence of a possible weak point for user privacy. The behavior of vendors is to not use them if not needed as it was observed in [2], where only the 10% of PRs analyzed use this specific type of data.

Important works, such as [30], underline the possibility that WiFi probe requests can give useful user's fingerprints, in particular the time features. These latter could be crucial for crowd-monitoring since they depend on vendor and model design. They highlight the behavior of each device model in sending single messages or bursts. There are three main fields in probe request time and burst characteristics:

- Inter-packet time: the time between messages inside the same burst.
- Inter-burst time: the time between two consecutive bursts.
- Number of packets within one burst.

These types of fingerprints can be an effective method for people tracking analysis. They give valid and non-intrusive data that could be employed inside frameworks remaining within privacy-conscious usage.

2.5.3 Computer network sniffers

The term *sniffer*, in the computer networks field, refers to the action of capturing the traffic flowing through a network, allowing an observer to discover and inspect the data exchanged between different devices or systems. Sniffers are commonly used for network troubleshooting, monitoring, and security analysis, providing valuable insights into network behavior, potential vulnerabilities, and performance issues. One of the most commonly used frameworks for network analysis is the Wireshark software suite [35], which permits capturing, filtering, analyzing, and monitoring all the information exchanges and traffic through a computer network. The most used file extension for these captures is *.pcap*, all the Wireshark

software suite is based on this particular file format. Tshark, a command-line interface tool, offers the option to use a lightweight version of Wireshark for network packet analysis. This can be useful since the majority of use cases foresee the use of small single-board computers. The most famous of these devices is the family of Raspberry Pi (Figure 2.7), developed by the Raspberry Pi Foundation [29]. This product has gained immense popularity as a cost-effective and versatile single-board computer, making it a popular choice for various projects and applications. Its compact size, low power consumption, and versatile Input/Output modalities make it an ideal candidate for network analysis tasks, including sniffing and capturing Wi-Fi traffic. The use of Raspberry Pi-based sniffers offers several advantages. Their small form factor allows for easy deployment in various locations, making them suitable for mobile crowd-monitoring applications. For example, they can be strategically placed in public areas, event venues, or transportation hubs to collect and analyze data related to crowd movements. Network sniffing can be done with appropriate software configuration and a WiFi dongle. This latter is a small device that allows a computer to connect to a wireless network, it is typically plugged into

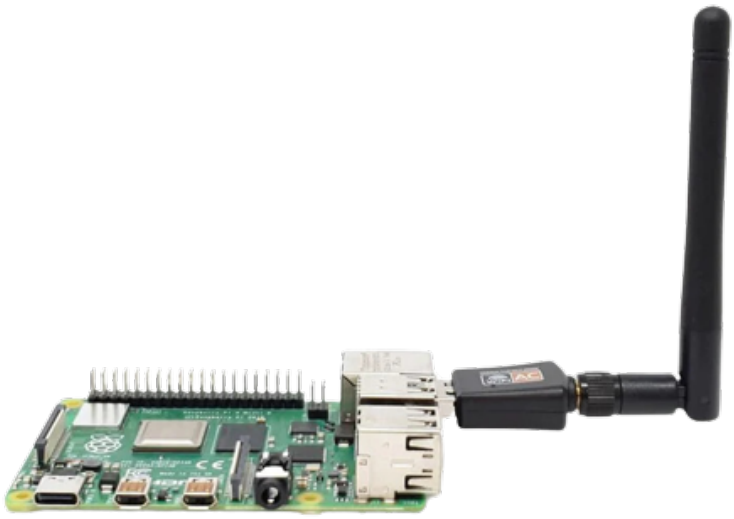

Figure 2.7. A Raspberry Pi board with a WiFi dongle.

a USB port of the device, providing it with access to Wi-Fi networks and wireless communication.

2.5.4 Systems for people counting

This section will give an overview of the main crowd-monitoring systems based on WiFi probe requests, starting from outdated methods, employed before the MAC address randomization, to the newest ones. These letters became harder than the previous ones due to privacy improvements applied by vendors. All the presented systems aim to estimate the presence of devices in a certain area by counting them. For all the described methods, the assumption is that each person owns a single device, simplifying the estimation of the number of people present to be equal to the number of devices detected. It is essential to acknowledge certain corner cases, such as the elderly and babies who might not possess smart devices and some individuals who may own multiple devices. However, this is a valuable assumption for cases like crisis management or urban traffic. In these situations, it is important to estimate a number that must be close, but not necessarily equal, to the real one. That is the reason why the one-device-one-person assumption can be kept as the most effective one.

Outdated frameworks

The adoption of MAC address randomization in smart devices has not always existed but started around 2014, as reported in [16]. From that year, gradually, all vendors began to embrace this method to guarantee more security features in their products. Before 2014, tracking people through PR sniffing was straightforward, both in case devices' addresses have the Globally/Locally bit set to zero or one. In the first situation, it is sure that the address is globally unique, on the other hand, in the second situation, the MAC address is locally administered, so there is no guarantee that it is worldwide unique. A possible solution for crowd-monitoring would include the usage of a list of MAC addresses extracted from the PRs. When a new address arrives there could be two situations:

- The MAC address is already registered in the list $\Rightarrow$ do not add it to the list.
- The MAC address is not already registered in the list $\Rightarrow$ add it to the list.

With the iteration of this simple method through all the capture, the number of devices detected could be easily obtained by counting the elements present in the list of MAC addresses. Given that, there is the possibility that some devices may use a locally administered address. In this case, two devices might use the same MAC address and this would lead to an underestimate of individuals. It can be said with a high degree of certainty that, even in very crowded environments, the probability of this happening would be very low. A possible implementation for

these outdated systems is reported with the pseudo-code provided in Algorithm 1.

Algorithm 1 Algorithm to compute the device counting before MAC address randomization

Require: *.pcap* file with the capture
  $MAC\_list \leftarrow empty\_list$
  for *packet* in *capture* **do**
    $MAC\_address \leftarrow get_MAC_address(packet)$
    if $MAC\_address$ is not already in MAC_list **then**
      $MAC\_list \leftarrow insert(MAC\_address)$
    end if
  end for
  $number\_of\_devices \leftarrow length(MAC\_list)$

After the introduction of MAC address randomization, these simple methods could no longer be used. This was an essential improvement for user privacy but, on the other hand, it has complicated a lot the possibility of monitoring through probe requests.

Conventional algorithms and de-randomization methods

The randomization process is not only exploited in WiFi probe requests but also in several other fields, for example in almost all the ones related to security and privacy problems. We can find randomization in bank transactions, message systems, and in the cybersecurity world. Some specific processes may be built to overcome the randomization and back to the original data. In the particular domain of probe requests, there have been efforts to address the randomization of these messages. The main focus is not to retrieve the original MAC addresses, since this would require a lot of computational power. The aim is to find a method that permits to understand which MAC addresses are more likely to be associated with the same device. The algorithms employed for this goal are based on simple computer science constructions, such as *if-then-else* statements, *loop cycles*, and *recursion*. The particularity is that the MAC address is not considered, but other fields are taken into account (i.e., the capabilities present in the other header's parts). With these methods, it is possible to group PRs sent by the same device with a certain accuracy. An example of work that uses a specific capabilities-oriented method is [20]. The system developed employs a recursive algorithm that explores the header's fields and classifies a probe request with a random MAC address as sent by a certain device. At the end of the process, it can count the number of devices present in the capture environment. This work has been used in [9] for the specific

crowd-monitoring task on bus lines.

The methods based on conventional algorithms and capabilities fields have demonstrated good accuracy in device counting, but they present some limitations. Firstly, the computational power required, especially where recursion is employed. In this case, if the message trace contains a huge number of PRs, there is the risk of saturating the memory of the devices where the algorithms run. Another problem is the accuracy evaluation since the captured messages do not provide the number of devices present in the environment (i.e., a ground truth) to ascertain the effectiveness of the developed method. In controlled scenarios, manually tallying the current devices and assessing the system's accuracy might be feasible. Conversely, in highly congested settings, such a process would be impractical.

Clustering methods

Clustering is a technique in the field of machine learning whose aim is to group similar objects or data points based on their inherent characteristics or attributes. The goal is to identify patterns, structures, or natural groupings within a dataset. This process is done without any pre-defined labels, for this reason, it is considered as a part of the *unsupervised methods* (i.e., machine learning techniques that aim to train a model without giving any information about elements' groups). Clustering helps in gaining insights, identifying similarities or dissimilarities between data points, and organizing large datasets into more manageable groups. An example of clustering schema is represented in Figure 2.8, where data are represented as points on the plane both before and after the clustering transformation. It can be seen

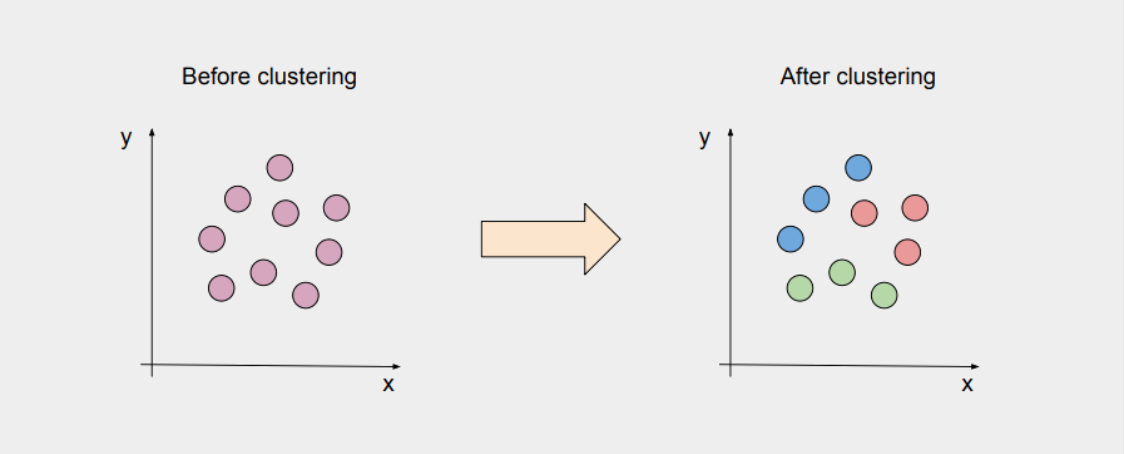

Figure 2.8. Data points spatial representation, before and after clustering labeling.

that data points before clustering are all represented with the same color since they are not labeled in groups, instead, after the clustering process, a label is assigned to each data point.

The process of clustering typically involves the following steps:

1. Data representation: the data is represented in a suitable format that allows the algorithm to understand the relationships between data points.

2. Choice of distance measurement: a distance metric is used to determine how close two data points are to each other. With this measurement, it is possible to aggregate data points in groups. Common metrics include Euclidean distance, Manhattan distance, and cosine similarity.

3. Choice of the number of clusters: determining the number of clusters to create. This can be specified beforehand (in this case referred to as *hard clustering*) or determined automatically by the algorithm (in this other case referred to as *soft clustering*).

4. Determine clustering algorithm: there are various clustering algorithms available, each with its strengths and weaknesses. Some popular clustering algorithms include k-means, hierarchical clustering, density-based clustering, Gaussian Mixture Models (GMM), and more.

5. Clustering process: the clustering algorithm is applied to the data, and it iteratively assigns data points to clusters based on their similarity, aiming to maximize the homogeneity within clusters and heterogeneity between clusters.

6. Evaluation: after the clustering process is complete, the resulting clusters are evaluated to assess their quality and coherence. Different evaluation metrics can be used depending on the nature of the data and the goal of the task.

Clustering can be used in crowd-monitoring systems with a very simple mechanism: group the probe requests as each one represents a device, and then count the number of formed groups to extrapolate the quantity of devices.

Among the various clustering algorithms, we can identify a specific family that is part of the soft clustering group: density-based clustering. Density-based clustering aims to identify regions in a dataset where data points are densely packed together, separated by regions of lower density. Unlike hard clustering algorithms, like K-Means, this type of clustering does not require specifying the number of clusters beforehand. Two popular density-based clustering algorithms are DBSCAN (Density-Based Spatial Clustering of Applications with Noise) and OPTICS (Ordering Points To Identify the Clustering Structure). DBSCAN may struggle with datasets containing clusters of varying densities and irregular shapes. OPTICS addresses these issues by creating a hierarchical representation of the data, allowing for

more flexible and robust cluster identification. In the context of crowd-monitoring systems, density-based clustering like DBSCAN and OPTICS can efficiently group probe requests based on their spatial density. Each formed group represents a device so that it is sufficient to count the number of clusters to extrapolate the total number of devices. Studies like [32] and [34] have leveraged these algorithms to achieve accurate device counting and crowd analysis in various scenarios.

In particular, in [32], the authors first divide probe requests with global MAC addresses from the ones with locally administered MAC addresses. Then they consider only the frames with locally administered MAC addresses and extract the main fields and capabilities of the frames. After that, they perform clustering methods (both DBSCAN and OPTICS) to assign the PRs to different groups. As the final step, the formed groups pass from a matching algorithm that tries to associate a group with one of the global MAC addresses encountered, if no matches are found, the group is left on its own.

Differently, in [34], authors decide to not consider the fields' values, but their lengths (i.e., the number of digits or characters that compose the values). Based on the previously described work, they initially categorize PRs into two groups: one based on globally administered MAC addresses and the other based on locally administered MAC addresses. Subsequently, they further group the PRs within the second category based on bursts. From these burst-based groups, some time information is extracted, like burst rate and inter-packet time. The group of PRs with global MAC addresses follows a counting method very similar to the one described before in 2.5.4. Instead, the burst-based groups are processed by clustering algorithms (such as DBSCAN and OPTICS) taking into account PRs fields and time information extracted.

The systems for crowd-monitoring based on probe request frames are more privacy-conscious than the others, but they still present some problems:

- The time required to compute huge mathematical calculations may be a lot due to the usage of machine learning algorithms like cluster-based ones. These solutions may require high-power computation systems that not always are present on the network sniffers or can not always be reached by them (with a wireless connection for example).

- The evaluation of these approaches is not simple, because the captured traces do not bring any sort of ground truth, i.e., the actual number of devices present in the environment is not embedded inside the capture done. A possible way would be to count the number of people present during the capture's time window, but it is clear that in very crowded environments this method would be not feasible. So the lack of ground truth is a huge problem in the evaluation processes for these types of systems.

Chapter 3

Probe request generator

One of the main contributions of this thesis is the development of a Probe request generator, built thanks to a solid knowledge of probe request messages and the 802.11 standard. This chapter will describe the motivations that led us to the development of this system, then it will provide all the technical details of the implemented solution. The chapter is divided as follows: Section 3.1 explains the reason why the generator has been developed, Section 3.2 describes the process of data collection necessary to build the generator, and finally, Section 3.3 reports the steps for the development of the generator of probe requests.

3.1 Lack of ground truth

One of the main problems in crowd monitoring systems that leverage probe requests is the difficulty in checking if the solutions implemented are effective or not. Once an algorithm is developed, the most important step is to evaluate the results obtained. Data used for tests must have a precise *ground truth*, i.e., a digit representing the number of devices inside the environment. The more closely the outcome provided by the framework and the ground truth are, the more successful the solution is. To evaluate the success of the system's performance is necessary to recover this number from the capture, but this task is not easy at all. One possibility to get the precious ground truth is to manually find the number of people present in the environment, with a person or a group of people that directly counts the single individuals. This method can only be applied in low-crowded situations, on the other hand, in cases such as big emergencies, outside a stadium, within the streets, or buildings of a city, it would be impossible to detect the actual number of devices. Another important aspect of having datasets with ground truth is the possibility of training machine learning algorithms. This latter is needed to find the best parameters of the mathematical model that compose the algorithm. Without a large number of huge datasets, it would be impossible to perform good training

for machine learning models. PRs are typically collected inside files with extension *.pcap*, where it is possible to read the packets' values with specific tools. These kinds of files represent the datasets for crowd-monitoring systems that leverage on PRs messages.

After a detailed analysis of message behavior, a Probe Request Generator has been developed, to tackle the problem of *Lack of Ground Truth*. The works done in [16], [30] and [9] provided a deep knowledge of the 802.11 standard, in particular for probe requests. To create realistic traces (i.e., files with .pcap extension) it is necessary to understand how PRs behave both in terms of fields' values and time characteristics.

3.2 Data acquisition

Section 2.5.1 discussed the main fields that compose a probe request and the time characteristics that can be extrapolated from a communication. As the first step in the creation of the generator, a series of captures have been done with different device models. A sniffer was configured to capture, within the network, only the probe request packets to analyze them in a series of .pcap files, one for each model. The method followed was equal to the one used in [30]: the captures were made in an isolated environment, where every device was turned off, except for the one under examination and the sniffer. Inside a single capture, the device switched its state in one of the three possible phases:

- Locked: it means that the device is secured or restricted from certain actions or access. This state is commonly used in mobile devices like smartphones or tablets when the user activates a security feature such as a PIN, password, fingerprint, or facial recognition lock.
- Awake: refers to the moment when the device is tapped by the owner, so when the screen lock is presented. It is not already actively used, but the user may be ready for the usage.
- Active: this state indicates that a device or application is currently in use and performing tasks or processing data actively.

Collecting packets transmitted by the device at various stages is crucial to observe how time-related characteristics evolve during different instances of device usage. After the collection step, all the .pcap files have been analyzed to extract the information relative to the following fields: VHT capabilities, Extended capabilities, and HT capabilities. Each of this information is collected inside a database where they are associated with the specific device model. Other fields, such as WPS, UUID-E, and SSID, are treated differently, the applied method will be explained

later in 3.3.6. After this first analysis, the .pcap files have been used to extract time and burst features such as inter-packet time, inter-burst time, and the number of packets per burst. Also in this case the features are associated with each device model inside the database.

Below is a table of devices tested and studied for the database population. For each one is reported the information about the type (i.e., if the device is a smartphone, a tablet, or a laptop), the vendor, the model, the operating system (OS), and the production year:

Type	Vendor	Model	OS	Year
SmartPhone	OnePlus	Nord 5G	Android 11.0	2021
SmartPhone	Samsung	Note 20 Ultra	Android 12.0	2020
SmartPhone	Apple	iPhone 11	iOS 15.0.1	2019
SmartPhone	Xiaomi	Redmi Note 8T	Android 10.0	2019
SmartPhone	Huawei	P9 Lite	Android 7.0	2016
SmartPhone	Apple	iPhone 6	iOS 12.5.5	2014
Tablet	Apple	iPad 8	iPadOS 14.8.1	2020
Laptop	Lenovo	ThinkPad X13 Gen1	Windows 11	2021
Laptop	Apple	MacBookAir M1	macOS 12.1	2020
SmartPhone	Apple	iPhone 7	iOS 15.2	2016
SmartPhone	Apple	iPhone 11	iOS 16.3.1	2019
SmartPhone	Apple	iPhone 13 Pro	iOS 16.3	2021
SmartPhone	Apple	iPhone 14 Pro	iOS 16.4	2022
SmartPhone	Xiaomi	Mi9 Lite	Android 10.0	2020
Laptop	Apple	MacBookPro	macOS 11.6.2	2015

Table 3.1. Devices Examined for Database Population

Database creation permits to have the necessary setup information, it represents the starting point of the system development. It is important to remember that the generator could be extended with the addition of new models inside the database, this may be an important part of some future works.

3.3 Generator state machine

The probe request generator structure is a state machine. This latter, also known as a Finite-State Machine (FSM), is a computational model used in computer science and engineering to represent the behavior of systems that can exist in a finite number of states. It is an abstract machine that transitions from one state to another in response to external inputs, internal conditions, or a combination of both. A state machine is characterized by the following components:

- States: distinct conditions or configurations that a system can assume at any given moment. Each state encapsulates a particular behavior or operational mode, delineating how the system will respond and function under specific circumstances.

- Transitions: define the conditions under which the state machine moves from one state to another. Transitions are triggered by events or input signals, and they determine the next state of the system.

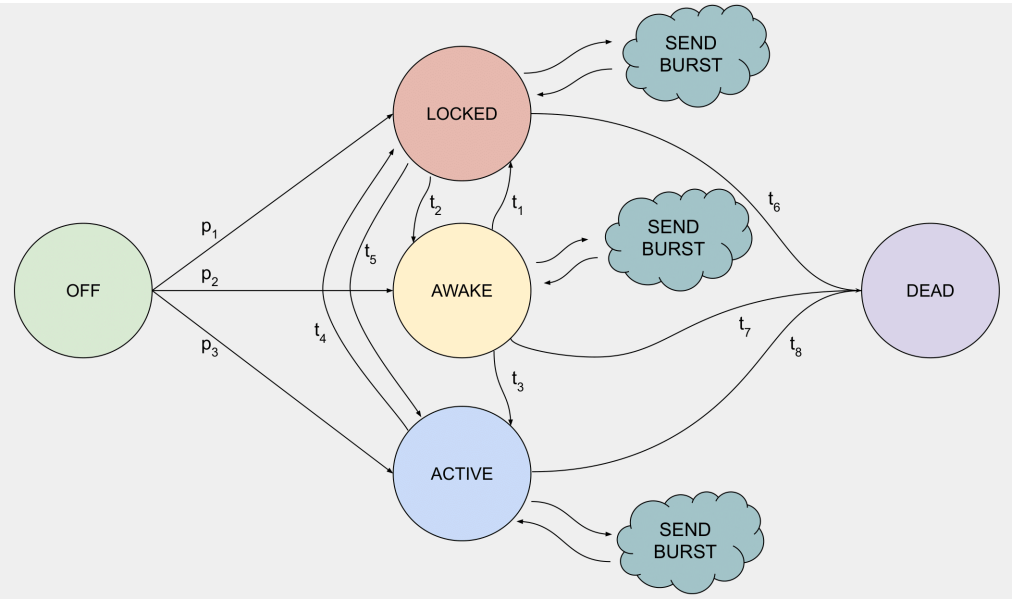

Figure 3.1. Finite-state machine of the probe request generator.

- Events: external stimuli or signals that cause the state machine to transition from one state to another. Events can be user inputs, sensor readings, or any other trigger that affects the system's behavior.
- Actions: represent the activities performed when moving from one state to another, they are linked with the transitions. They define what changes occur in the system's state or the environment during a transition.

Starting from this computational model, the PRs generator was developed following the environmental conditions and the system characteristics regarding probe request sending and sniffing. Every device is meticulously represented and simulated, encompassing the various states it can adopt and the corresponding actions it can execute. A schema of the finite-state machine created for building the generator is reported in Figure 3.1. Each part that composes the diagram will be explained in the sections below.

The whole generator implementation has been done with the Python programming language [27] and with some associated libraries that will be analyzed later.

3.3.1 Environment characteristics

A realistic simulation of probe request messages requires the construction of traces as captured in a real-world scenario. Packets built and sent by devices inside the simulated network are saved in files with extension *.pcap* to be processed later with the most important packet analysis systems (e.g., Wireshark). Throughout some parameters inside the simulator it can be possible to build a realistic environment for different use cases:

- Script time: permits to set the amount of real-time the simulation lasts.
- Average device number: the number of active devices, on average, inside the environment.
- Average permanence time: the average period during which a device stays operational before powering down or exiting from the given environment.

It is important to notice that the simulation and script times are unequal. So it is possible to set a simulation of some minutes and have the time window of captured packets of some hours. The difference between the two times depends on the number of devices present and simulated within the environment. With a small number of items to manage, the simulation time can be much higher than the script (i.e., real) one, and vice versa in more crowded situations.

3.3.2 Time features management

Each device analyzed during data acquisition has been saved in a database. Each time and burst feature (i.e., inter-packet time, inter-burst time, and the number

of packets per burst) are not expressed by one number only, but with a series of numbers and probabilities. This approach enables the assignment of specific weights to individual values during the process of selection. In particular, a Python library called *Numpy* [21] offers the possibility of doing a random choice from a set of values by associating each of them with a weight (in our case weight is the probability registered, and associated to the value). With this approach, the system can faithfully replicate the device's time and burst behaviors in sending probe requests. For instance, consider the inter-packet time of the Apple iPhone 6 during the locked phase. An illustration of the pair comprising a value and weight is as follows:

0.02:0.833 - 0.06:0.167

The device is expected to have around an 83% chance of displaying an inter-packet time of 0.02 seconds, while there remains a 17% probability of having an inter-packet time of 0.06 seconds.

3.3.3 Event list

Each device can be in different states and can undertake different actions, some of them can bring it to a new state. All the actions that a device can do are managed with a priority queue that is named *Event List*. A priority queue in computer science is a particular type of data structure that ensures that elements with the highest priority (this depends on the specific task) can be accessed and removed quickly. The main cases where it is used are the ones where tasks or items need to be processed in order of their importance. In this specific situation, the priority key is the execution time of the events. Each action has a specific time when it has to be managed and then executed. The timing is set inside the generator based on the current situation and the information taken from the models' database. The possible events inside the queue are listed below:

- *CreateDevice*: event that creates a new device. The initial phase (i.e., locked, awake, or active) of the device is decided with an experimentally chosen probability distribution (for more details see Section 3.3.5). The same event is responsible for the creation of other related events, such as *DeleteDevice*, *ChangePhase*, *CreateBurst*, and *CreateDevice*. To determine the vendor and model of the device to generate with this latter event, a specific website was consulted [18]. This resource provides regularly updated statistics on device sales and usage.
- *DeleteDevice*: event that deletes the device from the environment.
- *ChangePhase*: event that chooses the next device phase, based on the aforementioned probabilistic distribution and the current one. Then the next events *ChangePhase* and *CreateBurst* for the device are scheduled.

- *CreateBurst*: creates several events of type *SendPacket* according to the device number of packets per burst value, obtained from the generator's database. Then a new event of type *CreateBurst* is created.

- *SendPacket*: it contains the built probe request packet and saves it to the output .pcap file.

When a new event is created, the execution time is set based on the specific task (e.g., the consecutive events *SendPacket* are spaced by the inter-packet time chosen, while two *CreateBurst* events are spaced by the inter-burst time). Upon the creation of a new event, it is added to the queue. Following this, the event list is sorted based on the execution times of the events, positioning the one with the nearest execution time at the beginning of the list. An example of how the event list works is reported in Figure 3.2. It is important to specify that each event inside the list has its execution time, reported in round brackets. Although some of these values coincide with those in Figure 3.1, it is important to note that they represent different events despite all being related to execution times. They are all execution times, but they refer to different events. In this case, the generator has created a new event of type *CreateDevice*, with execution time equal to *t3*. The event list must insert the new item in the right position according to the other execution times. Since in the example *t3* is greater than *t2* and smaller than *t4*, the new event is placed between the third and the fourth ones. After that, the list continues to be processed by the system and the next event handled will be the *CreateDevice* with time *t1*.

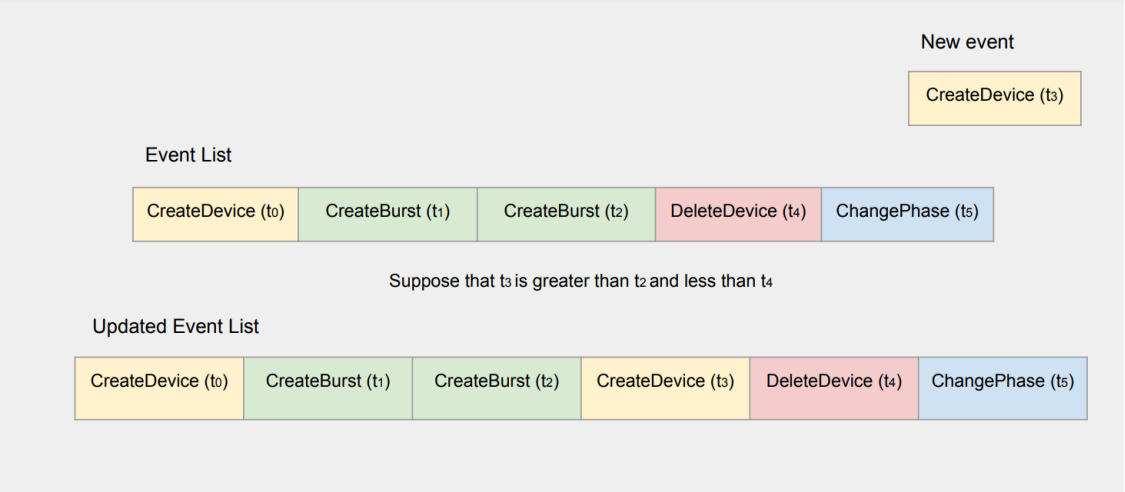

Figure 3.2. Event list of the probe request generator.

Event list cleaning

The occurrence of the *ChangePhase* event not only triggers a transition of the device's phase but also generates a new event of the same type, along with a new *CreateBurst* event. This latter will start a new sending packet chain, while the old one (derived from the past phase) is still inside the queue. For this reason, it is necessary to clean the event list every time a *ChangePhase* event is handled. In particular, it is sufficient to remove from the queue the device's *CreateBurst* events, apart from the one that has just been created by the *ChangePhase* one.

3.3.4 New device creation

To replicate use cases with utmost realism, it is necessary to account for the dynamic nature of individuals entering and exiting the captured area. In this context, the two main parameters to handle are the average permanence time and the average number of devices. By configuring these two values, it becomes feasible to plan the upcoming event of the *CreateDevice* type, thus creating a sort of frequency for generating the next device creations. The execution time for the next *CreateDevice* is not fixed but is calculated when a device creation event is needed. The mathematical law employed is Little's Law [15]:

$$L = \lambda W \tag{3.1}$$

Where:

L represents the average number of customers in a system at any given time (i.e., the average number of devices present in the environment).

λ represents the average arrival rate of customers into the system (customers per unit of time).

W represents the average time a customer spends in the system (i.e., the average permanence time).

In other words, the average number of entities in the system is equal to the product of the average arrival rate and the average permanence time. Little's Law is a fundamental principle in queue theory, a branch of operations research and applied mathematics that deals with the study of waiting lines (queues) and the analysis of systems involving entities waiting for service. Following this law, we can schedule the next device creation and maintain a roughly constant average number of devices in our generator environment. The death of the devices (i.e., the creation of the event *DeleteDevice*) has always the execution time equal to the average permanence time set as a system's parameter (as seen in 3.3.1). The main consequence of Little's law is the possibility of having a controlled environment, where the number of devices is aligned to the parameter set as desired. As observed, the timing for death and creation of devices are closely related to the inherent

dynamic dictated by the law, making it a key tool for keeping the system well-behaved and under control.

An important step in the creation of new devices is the decision of which model to create inside the environment. Each model inside the database is associated with its vendor so that it is possible to select the best-selling brands. The information about the market trends has been taken from a specific website [18]. Data collected from this source are then used every time a new device has to be created inside the generator. Specifically, as demonstrated in section 3.3.2, the Numpy library [21] provides the capability to make a random selection from a set of values while assigning each value a weight. These weights are sourced from the referenced website, enabling the recreation of a more realistic environment.

In Figure 3.1 the labels *t6*, *t7*, and *t8* represent the scheduled time for the event *DeleteDevice* which puts an end to the activity of a device. Given that the latter can be stopped at any phase it is in, the execution times are reported on three different arcs, even if death naturally only occurs once and its termination scheduling is established when the device is created.

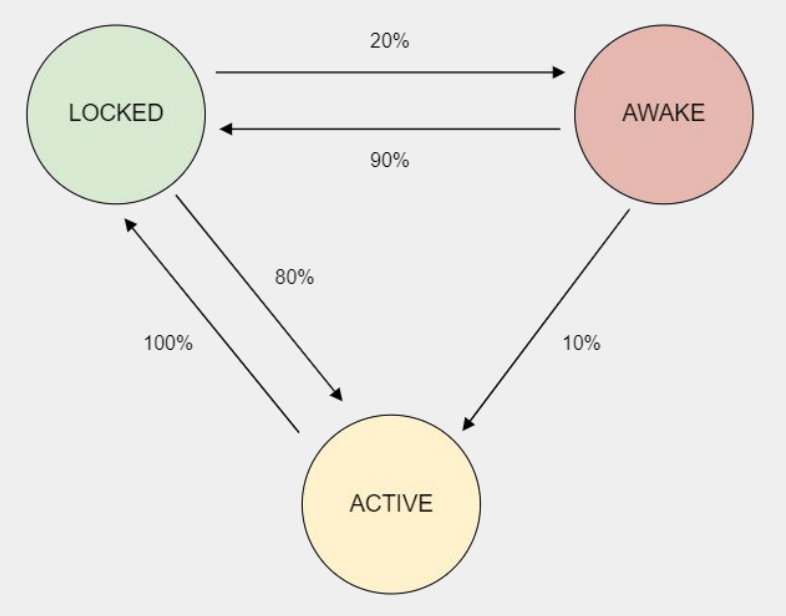

Figure 3.3. Probability transition values in the generator basic configuration.

3.3.5 Device phases

A device can send probe requests in all three main phases: locked, awake, and active. For this reason, time and burst features have been collected separately for each phase (as seen in 3.2). In addition, it is important not only to know how a device acts within a specific phase but also when it switches from the current phase and the next one. It is necessary to set a timing for each phase and a likelihood of moving to one phase rather than another. All of these tasks have to be managed within the generator, in particular, they are administered inside the handling of *ChangePhase* events.

The duration of each phase is controlled by introducing a new event called *ChangePhase* into the queue. This event is scheduled to execute at a time calculated as the sum of a base time (equal to the current time when the event is generated) and a specific number of seconds, equivalent to the phase's predetermined duration. The objective is to simulate human behavior, so the duration allotted to each phase is not fixed but instead follows a probability distribution. Specifically, a negative exponential distribution is employed for this purpose. This particular mathematical function possesses the characteristic that, with a roughly consistent rate of occurrence of the referenced event, it serves as a reasonable approximation for the time until the next event takes place.

The transition between phases for a device occurs quite frequently, which is why this distribution is chosen. The Python standard library's *random* [28] allows for the generation of random values adhering to a negative exponential distribution, with the possibility to set the average of the distribution. These average times are contingent on the specific use case and context in question, underscoring the importance of adjusting them as needed.

For instance, in a train station scenario, it is more likely that devices remain in the active phase for longer durations since people use them while awaiting trains. Conversely, in other scenarios, such as traffic monitoring on a street, it is more probable for devices to remain in the locked phase for extended periods. In the generator, estimates of the average phase durations have been initially set to 5 minutes for the locked phase, 30 seconds for the awake phase, and 3 minutes for the active phase.

A probabilistic model has been developed to represent the chances of moving from the current phase to various other phases, presenting a spectrum of probabilities for these transitions. Moreover, it should be noted that these probability values can vary among different usage cases, emphasizing the need for precise configuration. In the default setup of the generator, the values are established as depicted in Figure 3.3. In particular, the arcs represent the transitions and the labels on them report the chances to pass from one phase to another.

In Figure 3.1 the labels *t1*, *t2*, *t3*, *t4*, and *t5* represent the execution times of the *ChangePhase* events, when the device passes from one phase to another. The labels *p1*, *p2*, and *p3* are the chances that a device will start its life in one phase rather than another. In the initial implementation of the generator, these parameters were set to the following values: 35% for commencing in the locked phase, 15% for commencing in the awake phase, and 50% for commencing in the active phase. It is important to note that these values can be adjusted to accommodate various environmental conditions and requirements.

3.3.6 Probe request creation

The *CreateBurst* event has the main function of creating two other types of events:

- A certain number of *SendPacket* events all interspersed with time intervals equal to the device's inter-packet time in the specific phase.

- A new *CreateBurst* for the next burst creation that will be executed in a time interval equal to the device's inter-burst time in the specific phase.

The number of *SendPacket* created is equal to the number of packets chosen to compose the burst, this number can vary according to the probability distribution of the number of packets per burst, specific for each device. Each one of the *SendPacket* events has the task of generating a probe request packet with certain characteristics. Notably, the first *SendPacket* event of the series, has the same execution time as the *CreateBurst* event (i.e., the first packet of the burst created has the arrival time equal to the *CreateBurst* execution time). A useful Python library for building network packets is *Scapy* [31]. This tool permits to be compliant with the 802.11 standard so that building network messages becomes easier. Crowd-monitoring frameworks that exploit the use of probe requests, focus the attention on the *header* section of the transmitted messages and in some cases on a part of the message *payload* (such as the list of SSIDs). In networking, the header refers to the initial segment of a data packet, containing essential information about the message itself and its intended destination. In the context of probe requests, this header typically encompasses different features. On the other hand, the payload refers to the main content of the message, carrying the actual information that users need to transmit. Studies on PR frames (such as [16] and [30]) have demonstrated that nowadays the sequence number management in the messages has a precise schema. In particular, the first packet of each burst has a random sequence number (untied from the previous packets), and the following frames of the same burst have sequence numbers that progressively increase by one.

The following paragraphs will discuss the management of the other fields of the probe request header and the SSID.

MAC address

Each device inside the generator's database has a flag that indicates if it does the MAC address randomization. If the device does not implement randomization, its MAC address is built by taking the specific vendor's OUI for the first 24 Bytes, and for the other ones it uses random Bytes. It is important to underline that each packet sent by this device will have the same MAC address. For models that use randomization, every time a burst is created, a new random MAC address is used for that burst (i.e., for all packets inside the burst). This random MAC address has always the Globally/Locally bit set to 1, so it is a locally administered address, and the 8th bit of the first octet is set to 0, meaning that it is sent in *unicast* form. This latter transmission method involves directing the data (in this case would be the response of the found access point) specifically to a single intended recipient within the network and not to a group of systems as in the case of *multicast*. Every packet is transmitted with a destination MAC address of *FF:FF:FF:FF:FF:FF*. Frames with this specific destination address are sent to all hosts within a particular network domain and are commonly referred to as *broadcast frames*.

802.11 main fields

Probe request headers contain a lot of fields, some are different between models, others depend only on environmental factors while still others have roughly a fixed value. The ones that exhibit variations across different device models are collected during the process of data acquisition 3.2 and subsequently stored within the database. They are VHT capabilities, Extended capabilities, and HT capabilities. For the fields that depend on environmental conditions, such as the signal power, their values are randomly set inside the packets. Finally, for the ones that never change across different PRs, a default value is always used.

Special fields: WPS, UUID-E and SSID

Probe requests occasionally contain the WPS with the UUID-E in the header and the SSID within the payload. Following the studies done in [2] it is clear that only a small part of messages have these three fields, in particular, the couple WPS and UUID-E appear only in the 11% of the analyzed cases, while for the SSID field, we do not have precise statistics. Hence, within the *CreateDevice* event, an extra choice is available when generating a new device: there is an 11% chance that the device will be assigned randomly generated WPS and UUID-E values. If these values are assigned to a particular device, they are subsequently included in the packet creation process, specifically within the *SendPacket* event, where they are placed in its packets' headers. The procedure for handling the SSID within the payload remains consistent: inside the *CreateDevice* function, a decision is made whether the device will display a roster of its SSIDs or not. In the affirmative

scenario, an SSID list will be added to the device. Given the absence of definitive data regarding the inclusion of SSID in a probe request, the likelihood governing the possibility that a device shows its SSID list can be manually adjusted. In the initial version of the generator, approximately 20% of devices reveal their respective SSIDs.

3.3.7 Messages collisions

One of the main problems in packet trace generation is the management of the messages' arrival times. Especially in a very crowded environment, the probability that two probe requests sent by different devices arrive at the same time to the sniffer could not be ignored. Inside the generator, the .pcap file would have two messages with the same sniffing time. In real systems, these situations are called *collisions* and are managed through specifically designed recovery processes or with network segmentation.

Inside the probe request generator, when created, each packet is directly written inside the .pcap file. To address the potential for collisions and ensure that synthetic packets are not lost, an additional feature has been integrated into the system. Specifically, when a new event of type *SendPacket*, is going to be added to the list of events, the other ones with the same type are examined to check for potential time overlaps. If no collision is detected, the event is normally added to the list. Otherwise, a mechanism is triggered to adjust their timing. In particular, every time a collision is detected, the execution time of the new event to be added is shifted forward by *0.001* seconds. This small value has been chosen to have the slightest possible influence on the packet's original scheduled arrival time. After that, the new event is normally inserted inside the event list. The cumulative time by which the new event has been advanced is provided as the return value. This facilitates the synchronization of the subsequent packets within the ongoing burst to the revised arrival time established for the preceding packet. Once the necessary time adjustments are made, the event is appended to the simulator's list of events.

Once these necessary time adjustments are made, the event is appended to the simulator's list of events. As mentioned previously, this list is sorted based on the execution times of the events, with additional logic in place to handle cases where two events occur with overlapping time intervals. Specifically, when a *SendPackets* event (event B) and a *CreateBurst* event (event A) overlap in time, event A will create a new burst with its first packet. This packet will be scheduled to overlap in time with the packet related to event B. Ordering event B after event A ensures that the queue first processes event A and its subsequent *SendPackets* events according to the previously described logic. Conversely, if the ordering were reversed, with event A placed after event B, the packet related to event B would be written to

the .pcap file without considering the future burst generated by event B. The first packet of that burst would be written to the .pcap file with the same arrival time as the packet from event A, even though event A is no longer present in the event list.

Chapter 4

Proposed crowd-monitoring frameworks

This chapter will present a series of frameworks developed to perform crowd-monitoring, exploiting different technologies such as neural networks and clustering algorithms. These frameworks were developed to find a valid replacement for the outdated methods that became useless after MAC address randomization. All the methodologies and details involved in the creation of these frameworks will be provided. Their primary suitability lies in scenarios where devices remain predominantly stationary within a specific area, rather than in environments characterized by the continuous movement of people in and out of the capture zone. The chapter is organized as follows: Section 4.1 explains the reasons for adopting the Euclidean distance metric in the algorithms, Section 4.2 contains a detailed explanation about the type of clustering used, Section 4.3 describes an attempt to help traditional clustering algorithms with a neural network, and finally, Section 4.4 presents the framework developed by merging clustering methods with information obtained from the PRs studies.

4.1 Euclidean distance

For each of the implemented crowd-monitoring methods, it was necessary to find a reliable method to evaluate the distance between different probe requests. For comparing the PRs, it is possible to utilize the data contained within their fields by applying the mathematical concept of Euclidean distance. Each probe request can be thought of as a point in an N-dimensional space, with its features defining its coordinates along each dimension. To measure the likeness or disparity between two probe requests within this feature space, the Euclidean distance metric is a viable choice. It is essentially the straight-line distance between two points and it is calculated using the Pythagorean theorem, extending the basic two-dimension

distance formula to N dimensions.

Mathematically, the Euclidean distance between two points P and Q in an N-dimensional space is given by:

$$EuclideanDistance = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2} \tag{4.1}$$

Where:

q_1, q_2, ..., q_n are the coordinates of point Q along each dimension.
p_1, p_2, ..., p_n are the coordinates of point P along each dimension.

Thanks to this metric, complex feature data can be evaluated with a measure of proximity so that the difference and similarity between messages can be quantified.

4.2 DBSCAN clustering

The advanced crowd-monitoring systems seek to classify probe requests into separate categories based on criteria related to device or model distinctions. Clustering is a set of machine learning techniques that groups similar data points together based on certain features or characteristics. These algorithms work by analyzing the patterns and relationships within the data to identify clusters, which are collections of data points that exhibit high similarity to each other and are distinct from other clusters. They follow unsupervised learning techniques, which consist of training models on unlabeled data. In particular, the algorithms do not have access to explicit target values or predefined categories for the input. Instead, they aim to uncover patterns and relationships inside the data sets. In the thesis, the focus was on a particular subgroup of these techniques: density-based clustering. This latter focuses on identifying clusters based on the density of data points in the feature space. Unlike some traditional clustering methods, density-based clustering is capable of handling noise effectively. This latter refers to data points that do not meet the density criteria required to be considered part of a cluster. The main components and parameters of these types of algorithms are the following:

- Epsilon (ϵ): is the radius of the clusters.
- Min samples: is the minimum amount of points to consider a cluster as such.
- Metric: the measure to calculate the distance between points.
- Core points: a data point is considered a core point if there are at least a specified number of other data points (this number is specified with the *min samples* parameter) within a certain radius (*epsilon*) of it. These are the central points of clusters.
- Border points: a data point is considered a border point if it is not a core point but falls within the radius of a core point.

- Noise points: data points that are neither core points nor border points are considered noise points or outliers.

The *Density-Based Spatial Clustering of Applications with Noise (DBSCAN)* algorithm was proposed in 1996 as a new density-based clustering method [7]. The basic concept is to group data points in a dataset based on their density distribution. DBSCAN does not require you to specify the number of clusters in advance, this is a key point since the number of present devices or models is not known in advance. Instead, it identifies densely packed regions and labels points as core points, border points, or noise points.

The workflow of the DBSCAN is structured in six steps:

1. Start with an arbitrary point in the dataset that has not been explored yet.
2. Find all the data points within ϵ-distance of the selected point.
3. If the number of points found is greater or equal to *min samples*, the selected point is labeled as a core point.
4. If the selected point is a core point, a new cluster is formed. All reachable points from the selected point (i.e., within ϵ-distance) are added to the cluster and they are labeled as border points. This process continues until no more reachable points are found.
5. The algorithm continues by selecting the next not-yet-visited point in the dataset, and the process is repeated from 2.
6. After all points are visited, any remaining unlabeled points inside the dataset are signed as noise points.

Figure 4.1 and Figure 4.2 represent an example of data points before and after the DBSCAN clustering process. At the end of the algorithm, core and border points are shaded with their respective cluster-specific colors, while noise points are depicted in black.

Due to its comprehensive set of machine learning tools and user-friendly APIs, the DBSCAN algorithm was implemented with the support of *scikit-learn* library [33]. This latter provides a reliable and well-optimized algorithm that can be included within Python code.

4.3 Decoupling neural network

Crowd-monitoring systems that exploit clustering methods, often base their work on the similarity between PRs sent by different devices. The problem is that probe

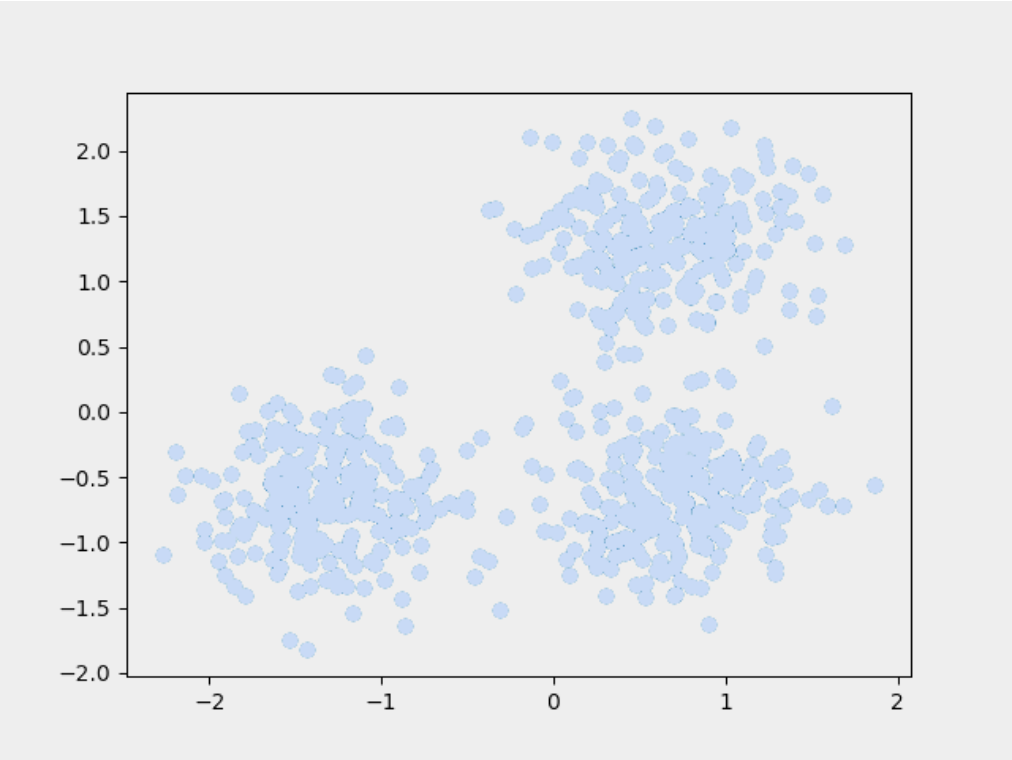

Figure 4.1. Data points before DBSCAN clustering.

requests are very hard to separate into groups based on device-specific features. Most of the time their fields are more effective when trying to separate them by models and not by a single device. Starting from this problem a method for decoupling PRs before the clustering process has been studied. A neural network was developed to process probe request fields and perform the task of pattern recognition. The final goal is to find a new representation for the messages that can be more effective in dividing them into groups, each one representing a specific device. The starting point of the structure creation was the concept of encoder-decoder neural networks. These are architectures designed to transform input data into a different and compressed representation, often referred to as a *latent space* or *encoding*. The encoding produced captures the most important features of the input data, which can then be used for various purposes like classification, generation, or further processing. The decoder part attempts to reconstruct the original input starting from this new representation provided by the encoder. If this reconstruction process brings an outcome close to the original input, it seems that the encoding is representative of the starting data. The reconstructed input and the original one are compared to evaluate the quality of the work done by the network. From this concept, the decoupling neural network was designed to find an inner

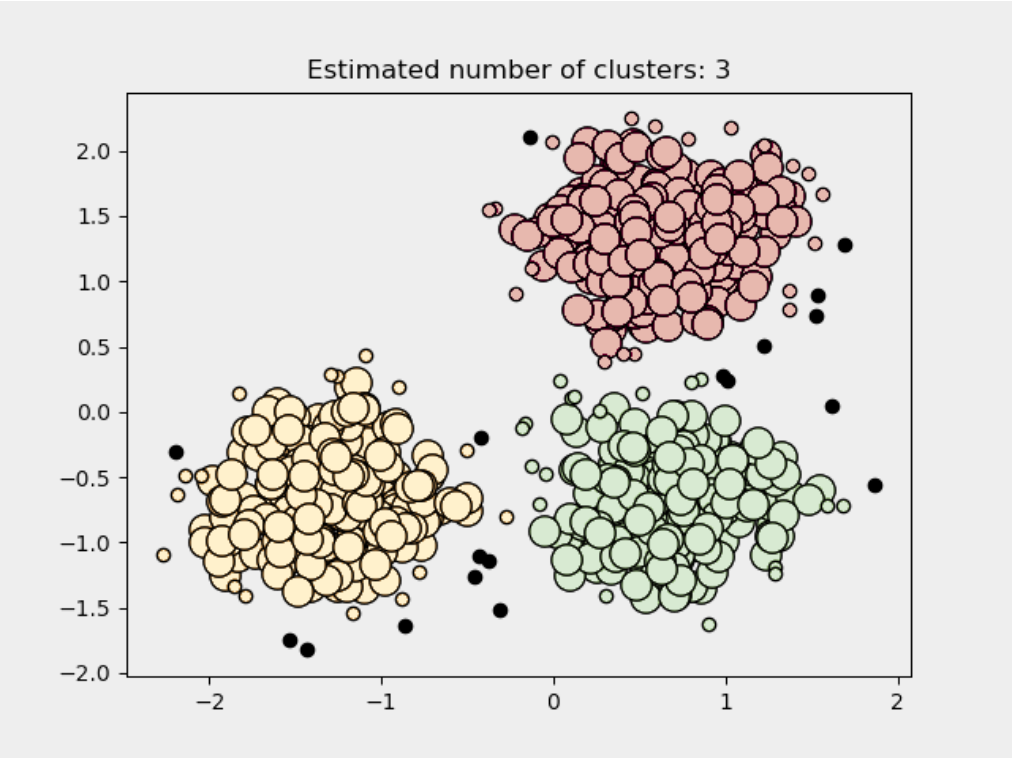

Figure 4.2. Data points after DBSCAN clustering.

representation of PRs that permits distinguishing a message sent from one device to another sent by a different device. The decoder part is not used since the goal is not to reconstruct a probe request starting from its latent representation so only the encoder part was developed. To evaluate the effectiveness of the encoding, a new method was implemented. This latter will be explained when the training phase will be treated. The developed neural network architecture is reported in Figure 4.3.

4.3.1 Training phase

During the training phase, the neural network aims to learn how to perform the pattern recognition described above. The learning process has the goal of setting the correct parameters for the specific task in the network's neurons. Probe request fields are processed to obtain a dataset where each row represents a probe request and each column represents the value of one field. After that, the tabular data is organized in *batches* (i.e., subsets of data) that are then passed through a series of *layers* (i.e., groups of neurons that do specific activities) that compose the encoder:

- Linear: a fundamental layer that performs a linear transformation on the input

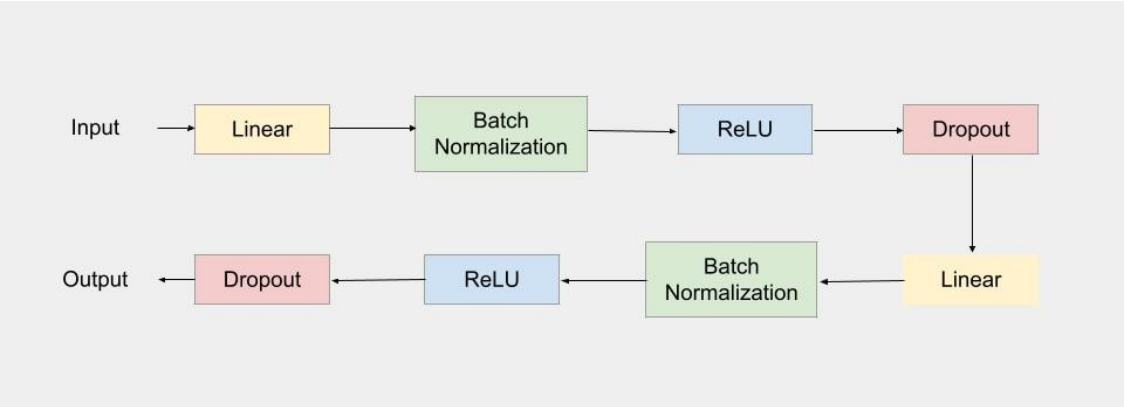

Figure 4.3. Encoder neural network architecture.

data, often followed by an activation function. It is commonly used for linear mappings and feature transformations.

- Batch normalization: this layer normalizes the activations of layers across a mini-batch of data, helping to stabilize and accelerate training.
- ReLU: the Rectified Linear Unit (ReLU) layer applies the ReLU activation function, which replaces all negative values with zero and leaves positive values unchanged. It is important because it introduces non-linearity to the network, making it more similar to natural processes.
- Dropout: is a regularization technique that randomly sets a fraction of input units to zero during each forward pass. This prevents overfitting (i.e., the scenario in which a machine learning model achieves outstanding performance on its training data but experiences a notably diminished performance when tested on unseen or new data) by reducing the reliance of the network on specific neurons and encourages the network to learn more robust features.

The sequential process, spanning from input data to the outcome, is commonly referred to as the *forward pass*. This series of steps collectively yields an encoding for the PRs. After that, the network has to know if the work done can lead to positive results for the task or not. The grade of success (or failure) is determined by a loss function, which serves as a cost function, quantifying the disparity between the predicted output of a model and the actual ground truth labels in the training data. The goal of training a machine learning model is to minimize this loss function, thereby improving the model's performance and predictive accuracy. After the evaluation is done with the loss function, the neurons' parameters are updated to minimize the function, this process is called *backward pass*.

It is important to set a good evaluation function for the model to make the neural network work well for the task. All the encoders produced inside a batch are compared with each other using the Euclidean distance paired with a threshold. This latter is decided at the beginning of the training phase and has the function of separating encodings in the space. In particular, given two encodings of two probe requests:

- If the probe requests come from the same device, the Euclidean distance should be under the threshold (i.e., the encodings are very similar because the two points they formed in the space are close to each other);
- If the probe requests come from different devices, the Euclidean distance should be over the threshold, meaning that the two encodings are properly different.

Starting from this assumption, every element within a batch is compared with the others using the Euclidean distance. Each probe request is labeled with the identifier of the device that sent it, so it is simple to know if two juxtaposed encodings come from the same device or different ones. A label is allocated to all the comparisons as outlined below:

- The comparison has the flag 1 if:
  - The two encoders derive from two PRs sent by the same device and their distance is *under* the threshold;
  - The two encoders derive from two PRs sent by different devices and their distance is *over* the threshold.
- The comparison has the flag 0 if:
  - The two encoders derive from two PRs sent by the same device and their distance is *over* the threshold;
  - The two encoders derive from two PRs sent by different devices and their distance is *under* the threshold.

This method aims to force the latent space representations to be similar (i.e., with Euclidean distance under the threshold) for probe requests that come from the same device and different (i.e., with Euclidean distance over the threshold) for probe requests sent by different devices. To create a proper loss function, all the flags obtained from encodings are compared with the real ones, which simply proceed from the ground truth labels. The two sets of labels are compared with a specific metric called *Binary Cross-Entropy Loss (BCELoss)* [4]. This function calculates the difference between the predicted probability distribution and the true distribution of binary outcomes, penalizing the model more for incorrect predictions and less for accurate ones. This loss function is particularly useful for tasks like binary

classification, where the goal is to categorize instances into one of two classes based on predicted probabilities. In mathematical form, the BCELoss can be expressed as follows:

$$BCELoss(y, \hat{y}) = -\left(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})\right) \tag{4.2}$$

Where:

y: represents the true binary labels, obtained from the ground truth.

$\hat{y}$: represents the predicted labels.

After the loss function evaluation, the training phase pursues the optimization of the neural network following the results obtained from the BCELoss. The optimizer adopted for this step is the *Adaptive Moment Estimation* algorithm better known as *ADAM* [13], which is commonly used in deep learning. In particular, this method is based on the fundamental optimization algorithm in deep learning, the *Stochastic Gradient Descent (SGD)*. The process begins with an initial guess for the model's parameters, which are typically initialized randomly or with small values. In each training iteration, the dataset is shuffled randomly to prevent the model from being biased by the order of data samples. Then, SGD selects a small, random subset of the training data known as a *mini-batch*. In the chosen mini-batch, SGD calculates the gradient of the model's loss function using all the network's parameters. This gradient represents the direction in which the loss function decreases the most. However, it is important to note that the gradient is calculated from this subset of data rather than the entire dataset, introducing a degree of randomness. The model's parameters are then updated based on the computed gradient. The step size in the parameter space is controlled by a hyperparameter known as the *learning rate*. It determines how much the parameters are adjusted in each iteration. The parameter update is performed for each mini-batch, and this process continues for a fixed number of iterations (referred to as epochs) or until a convergence criterion is met.

4.3.2 Test phase

For the test phase, the selected data must never been seen by the network because this ensures an unbiased evaluation of the model's generalization ability. The forward pass is performed, but no loss evaluation is done (in this phase there is no need for updating the network). After obtaining the encodings, the DBSCAN clustering algorithm is performed to group the messages. In particular, each group should represent a device, so that the total number of devices present in the system represents the number of clusters generated.

4.4 Clustering and time features of probe requests

In this section, an alternative crowd-monitoring system is introduced, which relies on clustering algorithms and incorporates insights gained from the probe request generator investigations. The flow-chart of the described method is reported in Figure 4.4

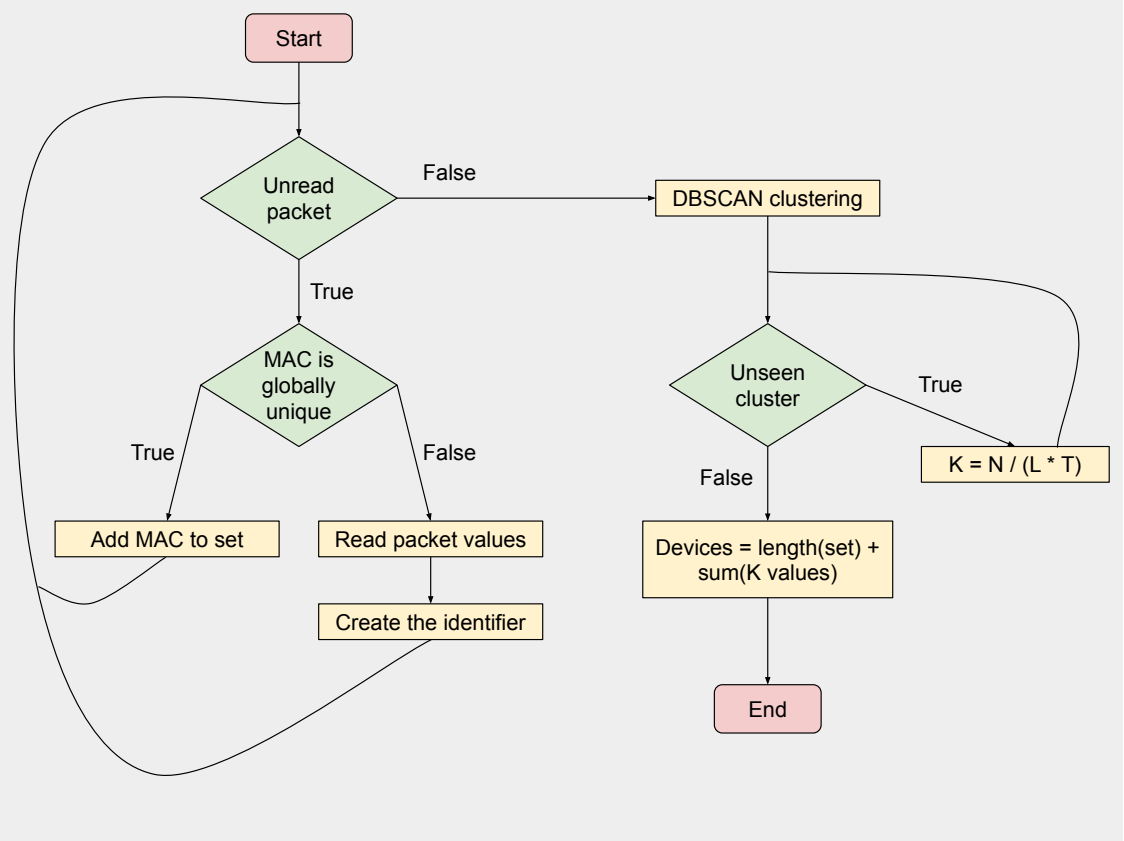

Figure 4.4. Flow chart for the developed framework.

4.4.1 Main capabilities and device models

Distinguishing between probe requests based on devices' features is challenging because header fields often contain information that is shared across all devices of a particular model. Clustering algorithms struggle to find a way to do pattern recognition. Even if all devices sent a similar number of PRs, it is a common occurrence for a part of generated clusters to have a substantial number of associated probe requests, while others contain only a small amount of messages. These results may

lead to the creation of multiple clusters, closely matching the ground truth in terms of the number of detected devices, but the message grouping is highly inaccurate. In subsequent tests and applications, the monitoring systems might struggle to accurately count the number of devices present.

Given that clustering is more effective when the objective is to distinguish probe requests solely by model. This framework focuses on leveraging model-oriented capabilities, in particular, the features that have been considered are VHT, HT, and Extended capabilities. While reading a packet from the .pcap file, the tools used present multiple subfields that compose one of the above-cited fields. The goal is to represent each capability with a number that can then be used as a dimension of probe request data points. The chosen technique to merge all the subfields referring to a specific field involves summing them to create an identifier. This results in a distinct numerical value being generated for each capability. Pseudo-code of the method to generate model identifiers is reported in Algorithm 2.

Algorithm 2 Algorithm to compute the probe requests model identifiers

Require: *.pcap* file with the capture
  $ids\_list \leftarrow empty\_list$
  for *packet* in *capture* **do**
    $HT\_counter \leftarrow 0$
    $VHT\_counter \leftarrow 0$
    $Extended\_counter \leftarrow 0$
    for $field$ in *packet* **do**
      if $field$ contains HT info **then**
        $HT\_counter \leftarrow HT\_counter + value$
      else if $field$ contains VHT info **then**
        $VHT\_counter \leftarrow VHT\_counter + value$
      else if $field$ contains Extended info **then**
        $Extended\_counter \leftarrow Extended\_counter + value$
      end if
    end for
    $ids\_list \leftarrow insert(HT\_counter, VHT\_counter, Extended\_counter)$
  end for

The result of this algorithm is the creation of an identifier for each probe request which recognizes its model. It is important to underline that the three capabilities are kept separate. The decision not to aggregate these values into a single identifier is intentional due to a potential concern. After the merging process, distinct messages sent by different models might share the same identifier, despite having different values for their three individual capabilities. Therefore, it is more prudent to keep the trio of capabilities separate.

4.4.2 Framework algorithm

All the probe requests analyzed must be inside the designed capture area, so they must be near enough to the sniffer. In some cases, messages that come from devices outside the desired region may be received. To solve this issue, the signal powers of PRs are compared with a threshold set to -70 dB. This value was chosen according to the experiments done in [30]. They demonstrated that this value permits filtering probe requests outside an area of about 10 meters radius around the sniffer. This threshold can be changed based on the desired capture area. Particularly, if this latter has to be extended, it is sufficient to set smaller values (e.g., -80/-90 dB) otherwise, greater values must be used (e.g., -60/-50 dB).

Before dividing probe requests by model thanks to clustering procedures, it is important to manage all the devices that do not use MAC address randomization (i.e., devices that provide directly their original MAC address). To distinguish globally unique MAC addresses from locally administered ones it is sufficient to see the Globally/Locally unique bit: if it is set to 1 it means that the address is locally administered (i.e., the device uses the randomization), otherwise, it is globally unique. Counting the number of these items is straightforward since it is sufficient to memorize their MAC address inside a Python *set*. This latter represents an unordered collection of unique elements. The key characteristic of a set is that it does not allow duplicate values. This makes it particularly useful when you need to work with a collection of items without worrying about repetitions. In this case, the set is perfect because, once a probe request with a globally unique MAC address is detected, it is possible to directly add the address to the set without processing the header fields. This approach ensures that duplicates are automatically avoided, allowing an easy device count. At the end of the .pcap file parsing it is sufficient to retrieve the length of the set to discover the number of devices that do not randomize the MAC address. For the other probe requests, a model identifier is calculated as described in 4.4.1.

Subsequently, these triplets become the data points that represent the probe requests. The entire set of these points is then provided to a clustering algorithm, in particular to the DBSCAN. At the end of the process, the aim is to have probe requests grouped according to the model of the devices that sent them. Before performing the clustering process, an additional check is done to verify that at least a certain percentage (set by default to 2% of the entire capture) of probe requests have a locally administered MAC address. This is done because it is important and makes more sense to apply the DBSCAN to a substantial part of messages received. In the case where this part of probe requests is under the threshold, only the messages with a globally unique MAC address are taken into account for the counting process.

DBSCAN algorithm assigns by default the label *-1* to every item that results

in the noise group. Since it may contain probe requests deriving from different models, this group is not representative of the crowd-monitoring method. The PRs associated with that label are automatically discarded. Starting from model groups formed with the clustering algorithm, it is necessary to count how many devices are present for every single model (i.e., for every single cluster). To obtain this estimate inside each group, it is important to consider the probe requests' rates. Each model sends probe requests with a certain frequency, depending on the phase in which it is. Since it would be very hard and not so accurate to divide clusters per phase, it was decided to consider only the average sending rate for a model. In situations in which crowd-monitoring would be performed in particular environments, where a certain phase would prevail in time over the others, the messages' rates separate for each phase may be useful. Hence, calculations have been performed for both the average sending rate across the entire capture and the average sending rate specific to each phase.

The formula exploited in this particular framework is the following:

$$N = K \cdot L \cdot T \tag{4.3}$$

Where:

N: total number of probe requests sent by devices of a certain model.
K: number of devices belonging to the model.
L: average probe requests' rate for the model.
T: time window in which the packets were sent.

This formula expresses the relation that elapses between these four parameters, the number of packets sent is directly proportional to the number of devices present, assuming a constant probe request rate and time window.

Particularly, in this case, the interest is in the parameter K so that the equation becomes:

$$K = \frac{N}{L \cdot T} \tag{4.4}$$

While parameters N and T are easily obtainable, the rate L has to be extracted from a source. A possibility would be possible to use 4.4 directly to the whole dataset of probe requests that use a randomized MAC address. All elements present inside the generator's database were tested to measure their probe request rates. These trials demonstrated that it is necessary to find a specific L value for each formed cluster since each model presented a significantly different rate.

Frequency association and counting by model

Each probe request has its model identifier which serves as a data point for the Euclidean distance inside the DBSCAN algorithm. All three capabilities used inside the identifier (i.e., VHT, HT, and Extended capabilities) bring information about

the *Throughput* of the system. This latter refers to the amount of data that can be successfully transmitted over a network in a given amount of time. The identifier can serve the dual purpose of grouping PRs into model clusters and assessing the proximity of these clusters to the generator's database models. The basic concept is that when two models share similar identifiers, there's a higher probability that their probe request rates will also be similar. Inside the generator's database, a certain number of models can be studied to obtain their identifiers and rates. This approach allows each created cluster to be compared with all the models in the database using their identifiers, enabling the identification of the most similar model, whose rate can then be utilized as the L parameter.

To collect these values, the probe request generator allows simulations with just one device from the database, making it possible to calculate the message rate associated with that specific model.

Particularly, a simple mathematical formula can be employed:

$$\text{Frequency} = \frac{\text{Number of packets in the .pcap file}}{\text{Simulation time}} \tag{4.5}$$

For each model present inside the generator's database, two types of information are computed: the triplet used for model identification and its probe request rates. All the data is stored within a file having the extension *.json*, structured similarly to the provided example:


```
"Apple iPhone14Pro": {
    "number": 1,
    "locked_rate": 0.08,
    "awake_rate": 0.44,
    "active_rate": 0.38,
    "mean_rate": 0.22,
    "probe_model_id": [0, 149, 16981]
},
"Apple iPhone13Pro": {
    "number": 2,
    "locked_rate": 0.11,
    "awake_rate": 0.67,
    "active_rate": 0.84,
    "mean_rate": 0.39,
    "probe_model_id": [0, 147, 16981]
},
"OnePlus Nord5G": {
    "number": 3,
    "locked_rate": 0.22,
    "awake_rate": 3.90,
    "active_rate": 0.54,
```

```
        "mean_rate": 0.43,
        "probe_model_id": [4052911066, 142, 972]
    },
    "Xiaomi Mi9Lite": {
        "number": 4,
        "locked_rate": 0.25,
        "awake_rate": 0.28,
        "active_rate": 0.24,
        "mean_rate": 0.26,
        "probe_model_id": [2442297858, 141, 843]
    },
    ...
```

JSON (JavaScript Object Notation) [12] is a lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate. JSON data is represented as key-value pairs, where keys are strings enclosed in double quotation marks, followed by a colon, and then the corresponding value.

With this new database of models, it is possible to find and exploit probe request rates to associate a parameter L to each model cluster. Before searching for a model inside the database, each cluster passes through a normalization process. In particular, not all the probe requests collected inside a cluster could have the same identifiers, they are expected to be similar due to the utilization of a density-based clustering algorithm within the framework, but there is no guarantee that they are identical. For this reason, inside each group, all identifiers are averaged to find a unique cluster identifier for the pairing process with the models inside the database.

Another important parameter in 4.4 is T. It represents the time window in which the probe requests are sent. As said before, our use cases are almost all related to situations where the devices are mainly stationary in the examined area. Therefore, the time window can be set as the total time of capture, equal for all clusters.

There are situations where a device may send a lot of probe requests during the capture window so that the ratio calculated in 4.4 becomes too high. To avoid clusters with an overestimated number of items, the relationship between the parameters N and T is monitored for each cluster. In particular, a parameter called *MAX_RATIO* is set properly inside the framework. The user determines the appropriate value for this parameter, taking into account the characteristics of the environment in which the capture is conducted.

This number serves as a threshold for an inequality performed for each cluster:

$$\frac{N}{T} < \text{MAX\_RATIO} \tag{4.6}$$

If the condition is satisfied, the Equation 4.4 is normally applied to the content of the cluster, otherwise, it seems that the cluster is probably formed by a small

number of devices that send a lot of messages. In this particular case, the number of items associated with the cluster is automatically set with a default value that must be small, like 1, 2, or 3.

The counting process is performed inside each cluster and the obtained results are summed together. The retrieved number is then summed again with the length of the set (the data structure used for messages with globally unique MAC addresses), and the outcome obtained is the estimated number of devices inside the environment.

Problem with multiple database matches

Triplets composed of VHT, HT, and Extended Capabilities are almost always different from each other inside the database. However, there are some cases where they coincide, in particular for some models of the same vendor. This poses a problem when a cluster identifies its closest match among these models, as the framework needs to make a singular decision. To solve this issue, it was decided to take all the rates of these selected models and average them to obtain the L parameter for 4.4. Since there is no certainty that the model represented by the cluster coincides with one or the other matches found, the average was considered a good choice.

Chapter 5

Experimental and numerical evaluation

This chapter will present a series of comprehensive tests about the two main thesis contributions: firstly the generator of probe requests, and then the crowd-monitoring frameworks. The effectiveness, efficiency, and performance of the proposed approaches will be proved with experiments, insights, and numerical outcomes. The exploration of empirical data and results will offer the necessary point of view into the real-world implications of these research efforts. The chapter is divided into two main sections: Section 5.1 for the probe request generator, and Section 5.2 for the crowd-monitoring frameworks. Each section will present the methodologies adopted to perform the tests and the results obtained from them.

5.1 Probe Request Generator

This section will explain the methodologies employed in the evaluation of the probe request generator described in chapter 3. All the experiments have been done with the support of the Python library *PyShark* [26] which provides a useful tools suite to analyze .pcap files and extract network packets' field values.

5.1.1 Methodology

The experiments done to validate the probe request generator have been conducted with a series of simulations inside a realistic environment with some characteristics:

- Fixed duration in terms of simulated time.
- Controlled environment with one only active device within the generator.
- Comparison between real traces and simulated ones.

- Separate analysis for each phase (i.e., locked, awake, and active).
- The observation window for all the statistics has been established at a duration of 30 seconds.

The generator's purpose is to replicate genuine traces by mimicking the behavior of a device in a manner consistent with real traces. For this reason, various metrics have been extracted from both the real traces (obtained from [30]) and the simulated ones. Specifically, these metrics are:

- The number of packets generated in the observation window.
- The time between packets inside the same burst (i.e., the inter-packet time).
- The difference between the arrival time of the last and first packets of the same burst (i.e., the burst duration).
- The number of packets per burst.
- The time between the capture of the last packet of a burst and the capture of the first packet in the next burst (i.e., the inter-burst time).

To compare real traces with simulated ones, it is necessary to measure the five values extracted from the .pcap files. In particular, two mathematical tools are used: the *mean* and the *coefficient of variation*.

The mean refers to the average of a set of numbers. It is often used to represent the central tendency or typical value of a data set. The mean is calculated by adding up all the values in the data set and then dividing by the total number of values:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{5.1}$$

To understand what is the coefficient of variation, it is first necessary to introduce the concept of *standard deviation*. The standard deviation is a statistical measure that quantifies the amount of variation or dispersion in a set of values. It indicates how spread out the values are around the mean:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}} \tag{5.2}$$

The coefficient of variation represents the ratio of the standard deviation to the mean of a dataset. It is often used to assess the relative variability of a data set, taking into account the scale of the data:

$$CV = \frac{\sigma}{\bar{x}} \tag{5.3}$$

The coefficient of variation is useful, especially in situations where it is necessary to compare the variability of different datasets or measurements that have different units or scales. It provides a more standardized measure that aids in meaningful comparisons and decision-making across different contexts. Regarding the validation of the generator, it enables the comparison of all five metrics, even if they have vastly different units of measurement and scales.

5.1.2 Numerical results

For the comparison between real and simulated traces, a specific device model was considered, the Apple iPhone 11. Figure 5.1 shows a simulated trace relative to only one simulation instance. Employing three distinct color bands, the differentiation of phases is visually represented. The yellow band designates the Locked phase, the green band pertains to the Awake phase, and the purple band characterizes the Active phase. Each row of the figure represents one of the five metrics used in the evaluation. The whole simulation represents a time frame of 40 minutes. It is important to highlight that the complete trace was generated in under 6 seconds, showcasing the scalability of the proposed method.

From the top down, the initial graph illustrates the count of packets generated inside the observation window. During the Locked phase, the device tends to transmit more packets at the start compared to the end. On the contrary, within the Awake and Active phases, there is an increase in packet count. The second plot demonstrates that the inter-packet time remains relatively consistent across all bursts. The values of the third plot are strictly correlated with the ones of the second plot and it is noteworthy to mention that the instances of zero values correspond to bursts containing only a single packet. Also for the fourth plot, the number of packets per burst constantly assumes fixed values. Finally, in the fifth plot, it can be seen that the inter-burst time has a high grade of variability as measured inside the real traces.

The validation is extended with more traces from our generator. For each one, the five metrics were extracted and the mean and coefficient of variation were computed. The outcomes of these comparisons are summed up in Tables 5.1, 5.2, and 5.3, reporting the real trace metrics and the simulated ones for all three phases. The measures inside tables are written with a series of acronyms, below is the list of them and their correspondences:

- PO: Packet Occurrences.
- IPT: Inter Packet Time.
- BL: Burst Length.

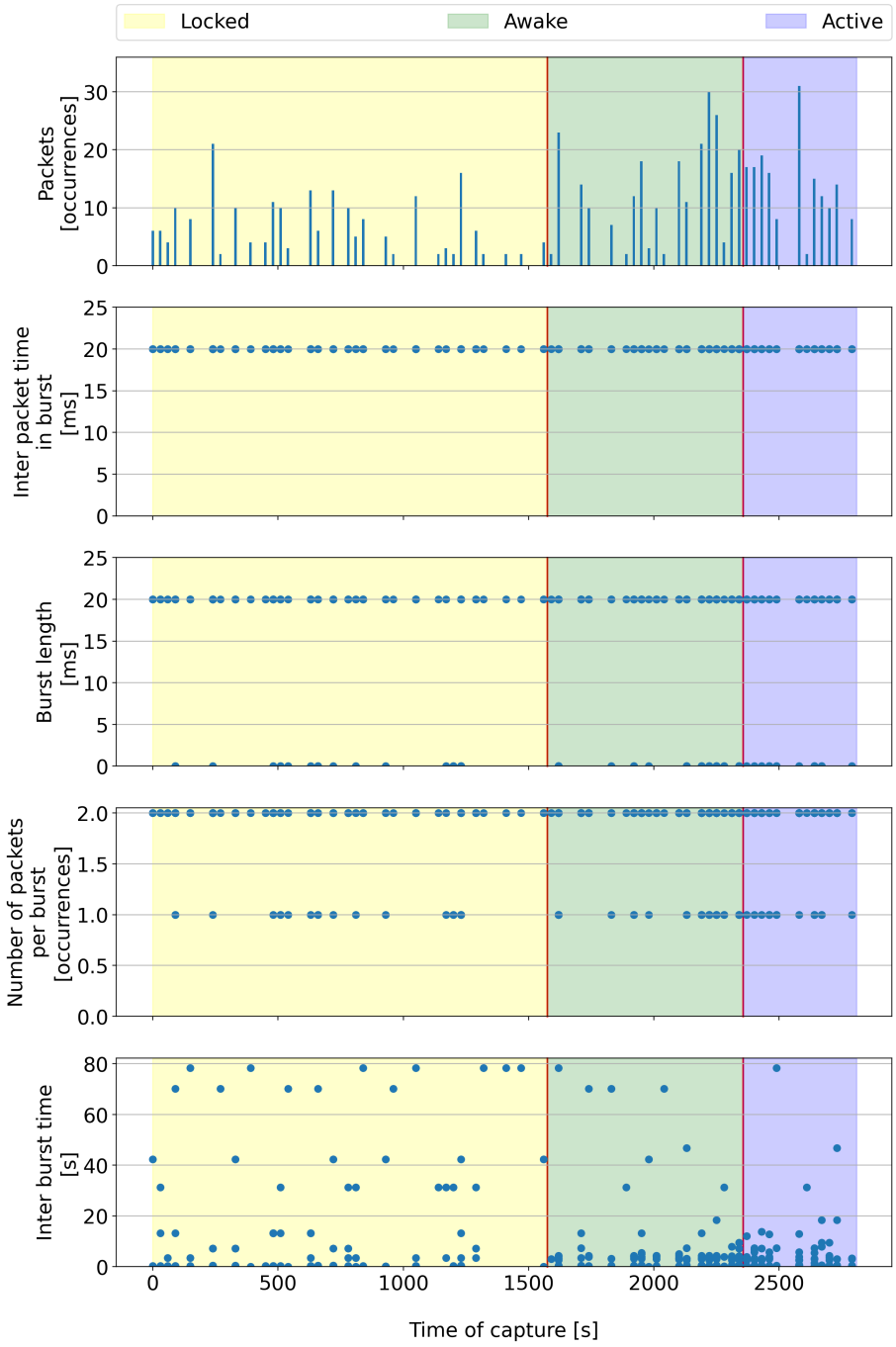

Figure 5.1. Emulated Apple iPhone11 trace as a function of time, for different device phases.

- PpB: Packets per Burst.
- IBT: Inter Burst Time.

The metrics are accompanied by the respective unit of measurement written in square brackets. Each of the four simulations conducted spans a simulated time of three hours, with one hour allocated to each phase. The choice to conduct these comprehensive simulations was deliberate, aimed at showcasing a robust correlation between the simulated behavior of the device and its real-world counterpart. This remarkable alignment owes its success to an internal simulator process that generates messages using empirical distributions gleaned from actual traces. If shorter traces had been used, it is possible that the reliability and consistency of the extracted metrics might not have been as strong, making it less indicative of the quality of management of the probabilistic distributions within the system.

From the results obtained, it is clear that the traces produced by the probe request generator and the real ones produce metrics that have very similar values for mean and coefficient of variation. This means that both the average value and its variation are successfully emulated.

Table 5.1. Locked phase Apple iPhone 11: (mean, coefficient of variation)

Metric	Real
PO [occurrences]	(3.11, 2.52)
IPT [ms]	(20.38, 0.02)
BL [ms]	(16.3, 0.52)
PpB [occurrences]	(1.8, 0.23)
IBT [s]	(16.49, 1.62)

Metric	Simulation 1	Simulation 2	Simulation 3	Simulation 4
PO [occurrences]	(3.42, 1.33)	(3.32, 1.36)	(3.29, 1.29)	(3.25, 1.34)
IPT [ms]	(20.0, 0.0)	(20.0, 0.0)	(20.0, 0.0)	(20.0, 0.0)
BL [ms]	(16.73, 0.44)	(16.5, 0.46)	(16.76, 0.44)	(16.85, 0.43)
PpB [occurrences]	(1.84, 0.2)	(1.83, 0.21)	(1.84, 0.2)	(1.84, 0.2)
IBT [s]	(16.07, 1.59)	(16.48, 1.57)	(16.74, 1.56)	(17.01, 1.55)

Table 5.2. Awake phase Apple iPhone 11: (mean, coefficient of variation)

Metric	Real
PO [occurrences]	(7.54, 1.46)
IPT [ms]	(20.33, 0.02)
BL [ms]	(17.94, 0.37)
PpB [occurrences]	(1.88, 0.17)
IBT [s]	(7.2, 2.32)

Metric	Simulation 1	Simulation 2	Simulation 3	Simulation 4
PO [occurrences]	(7.46, 1.15)	(7.74, 1.18)	(7.79, 1.13)	(7.42, 1.11)
IPT [ms]	(20.0, 0.0)	(20.0, 0.0)	(20.0, 0.0)	(20.0, 0.0)
BL [ms]	(16.68, 0.45)	(16.55, 0.46)	(16.66, 0.45)	(16.5, 0.46)
PpB [occurrences]	(1.83, 0.2)	(1.83, 0.21)	(1.83, 0.2)	(1.83, 0.21)
IBT [s]	(7.35, 2.26)	(7.06, 2.33)	(7.04, 2.3)	(7.36, 2.27)

Table 5.3. Active phase Apple iPhone 11: (mean, coefficient of variation)

Metric	Real
PO [occurrences]	(11.16, 1.06)
IPT [ms]	(20.27, 0.02)
BL [ms]	(16.6, 0.47)
PpB [occurrences]	(1.82, 0.21)
IBT [s]	(4.81, 2.41)

Metric	Simulation 1	Simulation 2	Simulation 3	Simulation 4
PO [occurrences]	(10.47, 0.89)	(10.85, 0.91)	(10.87, 0.89)	(10.71, 0.85)
IPT [ms]	(20.0, 0.0)	(20.0, 0.0)	(20.0, 0.0)	(20.0, 0.0)
BL [ms]	(16.4, 0.47)	(16.85, 0.43)	(16.62, 0.45)	(16.61, 0.45)
PpB [occurrences]	(1.82, 0.21)	(1.84, 0.20)	(1.83, 0.2)	(1.83, 0.21)
IBT [s]	(5.2, 2.39)	(5.08, 2.42)	(5.03, 2.38)	(5.11, 2.27)

Figure 5.2 presents an example of a generated packet from a simulation. Wireshark visualization tools permit to read the .pcap file contents. It is possible to control if the fields are present and were correctly set by the generator. An important verification is the existence of malformed packets inside the capture, that are signaled with a red error message. This happens in particular when the message's fields do not respect the protocol's standards, such as their values are too short or too long. The correctness of generated packets was checked so that Wireshark does not detect any malformed packets.

5.2 Proposed crowd monitoring framework

This section will present all the experiments done for the crowd-monitoring frameworks. The tests have been done with multiple datasets representing different environments. In particular, both the main frameworks were subjected to a strict comparison of each other and with an outdated system developed with the same methodology reported with Algorithm 1. Again, the section will present the methodologies employed for the experiments and the numerical results achieved by the systems.

5.2.1 Methodology

The procedure used involves conducting a series of tests to evaluate the performance and effectiveness of the frameworks in various scenarios or situations. These experiments are designed to mimic and represent different real-world usage scenarios for

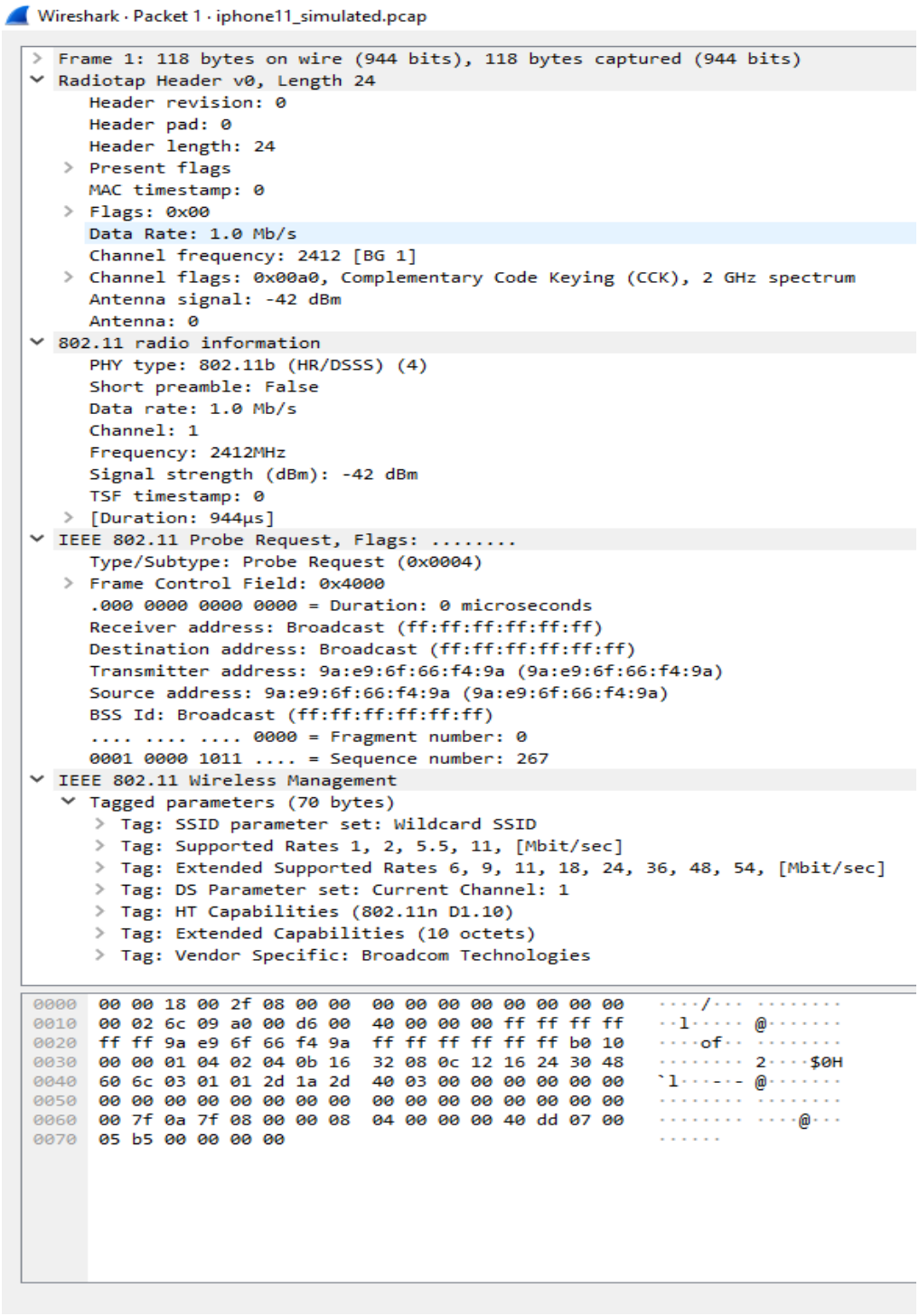

Figure 5.2. An example of a generated probe request packet.

the systems. The datasets employed cover a range of environments, starting from scenarios where there is only one device present, to others with multiple devices.

The models' distribution may be different, there can be situations where the majority of devices belong to few models as well as situations where models are more diversified. Another aspect to consider is the different sources of the analyzed traces. They can be from the generator or real captures. In this work, both types are used.

It is important to underline that the captures from the generator have the same types of devices stored inside the database presented in 4.4. So the matching process between formed clusters and the database is straightforward in this case since each cluster will find the exact model and not only one with the closest identifier. In real captures, there is certainly a higher variability of models so the matching process may be less precise. This is the reason why it was decided to measure the effectiveness of the frameworks not only with synthetic data but also with real captures. Below is an in-depth explanation of the methodologies employed for each of the used frameworks. Some of the methods presented require setting parameters to specific values, so for each test done, these values will be reported in the explanation.

Outdated framework

Methods used before the MAC address randomization required just a simple Python set to save the MAC addresses. Before the implementation of privacy-conscious techniques, each device sent its probe requests with the original MAC address, which is globally unique. It is sufficient to store all the captured addresses inside the set and retrieve the length at the end of the analysis. Since this data structure does not allow element repetitions, it is sure that all the single MAC addresses are stored so that it is very easy to obtain the number of devices present. The experiments done with these old techniques have been performed as described above, simply with the use of a set. The results obtained are compared with the ones obtained from the two frameworks to highlight the importance of developing these alternative solutions.

Decoupling neural network

The decoupling neural network has been trained with a simulation that comprises all the devices present inside the generator's database. The probe request encoding performed by the network reduces to a quarter of the original length of the message fields. This dimensionality reduction was chosen after some trials that demonstrated the possibility of having a smaller representation of PRs without losing information, but it can be arbitrarily set. The learning rate necessary for the *ADAM* optimizer is set to a default value of 1×10^{-4}. This parameter essentially

controls the speed at which the optimization converges to a minimum (or maximum) point in the loss landscape. Later we show how the network learns the task assigned very quickly, this is the reason why the learning rate has not been passed through a validation process. An important parameter to choose for the training phase is the threshold on which the algorithm works to differentiate the probe requests sent by distinct devices. Fundamentally, this parameter must be kept equal in the test phase. After conducting several trials to comprehend the values taken by the Euclidean distances between probe requests and how they change following the encoding transformation, the decision was made to establish the threshold at *3*. It has been shown that the network can effectively distinguish messages sent by different devices with a relatively small number of iterations. This conclusion has been validated through a training phase conducted using a dataset containing 55 devices of various models.

Figure 5.3 reports the accuracy in probe request differentiation over the training iterations. In particular, this metric is measured based on the distance between two PRs:

- If they come from different devices, they must have an Euclidean distance at least equal to the threshold.
- If they come from the same device, they must have an Euclidean distance under the threshold.

If the majority of juxtapositions between messages follow the above cases, the accuracy rapidly grows since it is measured as the percentage of successful comparisons. The graph illustrates the rapid increase in accuracy after a few initial iterations, followed by a relatively stable performance between 100 and 200 iterations. Figure 5.4 presents the trend of the Euclidean distance for both couples of messages from the same device and different devices. The blue line represents the distance between probe requests sent from different devices, while the green line shows the distance between messages sent from the same device. The red line stands for the threshold value. It can be seen that, after a few iterations, the algorithm can distinguish well message sources based on the training done with the threshold. Both graphs indicate that the number of iterations needed to achieve stability in the system's results trend is relatively low, especially when compared to the effort typically required during the training phase of neural networks for tasks such as computer vision.

In machine learning algorithms, a fundamental practice for dataset management involves the partitioning of data. Specifically, it is a common approach to split the dataset into two segments, with one comprising 80% and the other 20% of the entire collection. The first subset is used for training, while the second is for testing. It is important to not give the test data to the model before testing so that the

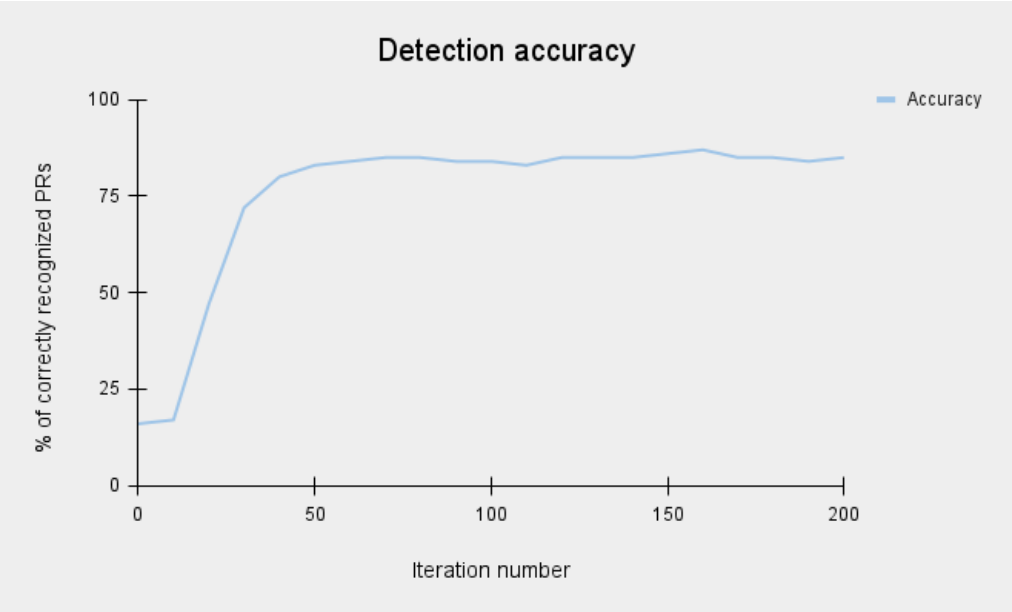

Figure 5.3. Accuracy in recognizing probe requests from the same device or not.

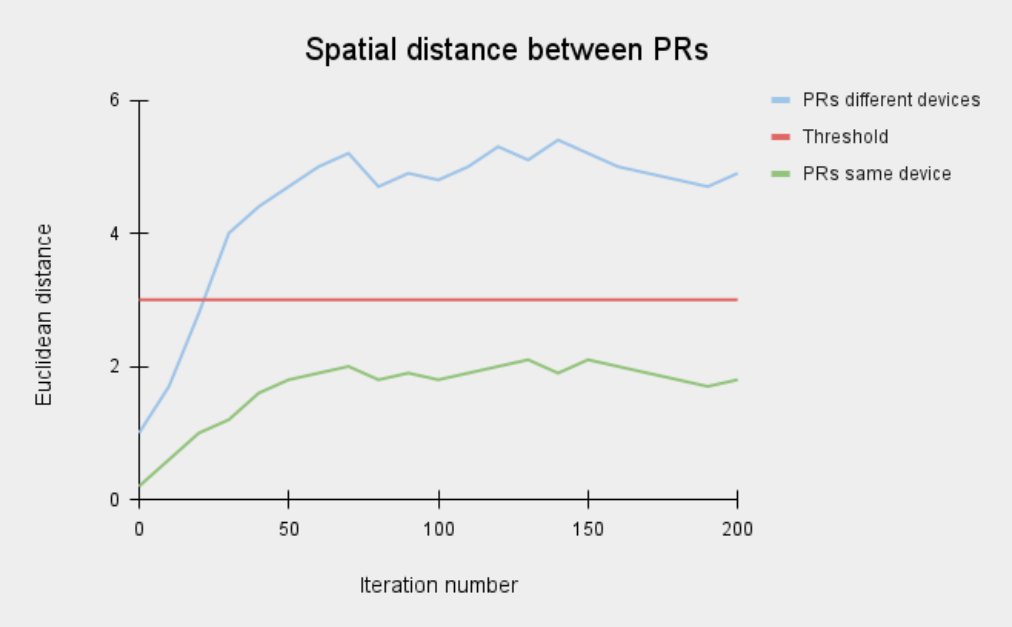

Figure 5.4. Probe requests' spatial distance during the training phase.

algorithm can not learn about these items and the evaluation can be done with unknown elements. For the decoupling neural network, the 80/20 approach (this is

the name by which it is known) was adopted, so the probe requests from the dataset used were divided into two groups. During the test phase, the neural network is used with the parameters learned during the training phase. The probe requests are transformed into encodings and then processed with the DBSCAN clustering method. To be compliant with the training phase, the parameters of the algorithm must be set properly. The *scikit-learn* library [33] permits easily to assign values to these parameters. They have been set as follows:

- Metric = Euclidean distance.
- Epsilon = threshold (i.e., 3).

Inside the DBSCAN definition, there is also the *Min samples* parameter whose value depends on how crowded is the environment and how many seconds the capture last. For example, when dealing with traces characterized by narrow time windows, certain devices, and, consequently, the formed clusters, might be linked to a limited number of messages. Therefore, within these smaller groups, the sample size may not be particularly high. This underscores the importance of adjusting this parameter to suit the specific use case.

Clustering and time features framework

The clustering framework uses also the DBSCAN algorithm for the set of probe requests that have a locally administered MAC address. In this case, the parameters are configured in the following manner:

- Metric = Euclidean distance.
- Epsilon = 4.
- Min samples parameter depends on the environment and the duration of capture, as discussed in the paragraph above.

After conducting a series of trials, the decision was made to set the *epsilon* parameter to 4. The primary objective behind these trials was to gain insights into the spatial relationships among the triplets composed of VHT, Extended, and HT Capabilities, as measured by the Euclidean distance metric.

There are two other parameters to set for this framework, MAX_RATIO and $DEFAULT\_VALUE$. The first aims to monitor the ratio between variables N and T so that it remains under a certain threshold. The goal is to avoid situations where the count would be wrong due to devices that send a lot of messages or when the time window is too small. The tests done have shown that for low or medium-crowded environments, the threshold for the ratio can be set to values close to 1, such as 0.8, 0.9, or 1 itself.

The *DEFAULT_VALUE* parameter determines the number of devices to assign to a cluster if the condition associated with the *MAX_RATIO* is not respected (i.e., the ratio between N and T is over the threshold). The value is chosen arbitrarily, but in most cases, it has been set to a small natural number, like 1.

Datasets used

The goal of these studies is to analyze the performances of crowd-monitoring systems under different environments. The datasets used for the tests are all .pcap files. Thanks to the generator of probe requests, it is possible to generate various situations. Some with more emphasis on having a few devices but many different models, some others with the opposite case, and still others with mixed conditions. The algorithms have been tested also with real .pcap traces, even if in these cases another problem takes over: the difficulty in having a precise ground truth. This latter was obtained with an estimate done by the person who collected data, based on the average number of people or devices believed to be there. Below is the complete list of datasets used during the experiments:

- Simulated traces:
  - *Dataset A*: 60 devices all belonging to one model only.
  - *Dataset B*: 6 devices each one representing a different model.
  - *Dataset C*: contains only one device.
  - *Dataset D*: medium-crowded situation, with 70 devices of various models.
  - *Dataset E*: large-crowded situation, with 120 devices of various models.
- Real traces:
  - *Dataset F*: contains only one device.
  - *Dataset G*: capture done inside a home, 9 smart devices detected.
  - *Dataset H*: contains the captures conducted within an anechoic chamber as part of the research detailed in [23].
  - *Dataset I*: comprises a collection of 10 two-minute captures performed within room 14 at Politecnico di Torino, on the first day of the 2023/2024 academic year.

As told before, each dataset represents a different environment with its characteristics. With these traces, it has been possible to properly evaluate the developed systems. *Dataset I* presents a "dynamic" use case, where the environment was captured a few minutes before the start of a lesson and continued recording for a few minutes afterward. This recording scenario allowed us to simulate the gradual entry of people into the room, reflecting a real-world situation. Starting from the

the beginning of the scenario, where a few individuals were already present in the room, and continuing over the next twelve minutes. Gradually, more people entered the room, and in the final three capturing windows, the lesson began, with only a small amount of additional individuals arriving.

5.2.2 Numerical results

This section presents a collection of tests done with the datasets presented above. Each of the developed frameworks was tested by inputting the .pcap files and obtaining the number of identified devices.

The outcomes of the experiments have been documented in Table 5.4, with each row corresponding to a particular series of tests conducted using a .pcap file that simulates an environment. The columns furnish various details, including dataset identifier, ground truth, the detected device count, and accuracy for each framework. The accuracy was computed with a methodology that takes into account the distance between the result and the ground truth. More specifically, the counting error was calculated as the absolute difference between the ground truth and the outcome, then the result was normalized by the ground truth. The accuracy can easily be retrieved as one minus the error obtained. The two equations for the calculus are reported below:

$$Err = \frac{|GT - R|}{GT} \tag{5.4}$$

$$Acc = 1 - Err \tag{5.5}$$

Where Err is the error, GT is the ground truth, R is the result obtained, and Acc is the accuracy. The value of the latter is reported near the number of devices detected in round brackets and the best result for each dataset is written in bold. To maintain the table simple to read, it was decided to assign to the frameworks a set of acronyms:

- Outdated: Outdated method.
- DNN + C: Decoupling Neural Network with Clustering.
- C + T: Clustering with Time features.

While Equation 5.5 provides a feasible metric for measuring framework performance, it overlooks a crucial aspect: the diversity in the number of devices within the environments. Some datasets have many devices, while others do not. *Dataset E* has over 100 simulated elements, while *Dataset C* has only one device. It is clear that an error of, for example, 1 device has a more pronounced impact in a less crowded environment but could be regarded as negligible in a highly populated one. That is the reason why it is important to not evaluate the accuracy on its own but coupled with the simulation context. *Dataset I* consists of several recordings, so

that the resultant accuracies from the tests performed are averaged and presented within the table.

Table 5.4. Crowd-monitoring results for different frameworks.

Dataset	Ground truth	Outdated	DNN + C	C + T
A	60	329 (18%)	50 (83.33%)	**52 (86.66%)**
B	6	709 (0.84%)	4 (66.66%)	**7 (83.33%)**
C	1	793 (0.12%)	2 (50%)	**1 (100%)**
D	70	5189 (1.35%)	**58 (82.85%)**	**58 (82.85%)**
E	120	6180 (1.94%)	107 (89.16 %)	**110 (91.66%)**
F	1	179 (0.55%)	3 (33.33%)	**2 (50%)**
G	9	45 (15%)	**14 (64.28%)**	**14 (64.28%)**
H	22	3194 (< 0%)	17 (77.27%)	**23 (95.45%)**
I	up to 121	overall 1.5%	overall 71.5%	overall **91.48%**

The outdated framework demonstrated its poor performance in all the tests done. Since almost all devices nowadays use MAC address randomization, counting the single addresses seen in the capture is not precise anymore. The other two methods have achieved good results in almost all cases. For *Dataset F* both the systems did not give excellent outcomes in terms of accuracy, and this is the perfect example of what was explained above, the importance of the context. *Dataset F* has one device only, so an error of 1 extra counted device drastically drops the accuracy down to 50%, even if an estimation of 2 devices instead of 1 should be considered outstanding.

Particular results derive from the use of the decoupling neural network. In some tests, the number of devices obtained is very close to the ground truth. Accuracy is an important strategy to measure the effectiveness of the frameworks, but there are cases where it is not the only metric to take into account. Another aspect to take into account is the composition of the formed groups, especially whether the majority of elements within each cluster are correctly assigned. For the DBSCAN results after the decoupling neural network, it was shown that the groups formed (i.e., the probe requests containers for each device) were very inhomogeneous. On

one hand, one aspect highlights substantial clusters that contain an unusually high number of messages compared to the actual quantity of probe requests sent by individual devices. On the other hand, there are smaller clusters with only a limited number of messages, which falls short of the genuine number transmitted by devices. This analysis was possible thanks to the information that the probe request generator stores and gives to the user during and after the simulation. The number of clusters formed to divide messages after the decoupling phase may be equal to the actual number of devices. However, this cannot be regarded as indicative of the framework performing well, as the content does not accurately represent the genuine distribution of probe requests.

Better results both in terms of accuracy and cluster nature were obtained from the DBSCAN clustering method coupled with the time features extracted from the domain knowledge. It has been demonstrated that forming groups to categorize probe requests based on the model, rather than individual devices, proves to be more efficient. The analysis of clusters was done and showed their genuine nature. Specifically, the algorithm accurately separated the majority of probe requests based on their device models, achieving a correct assignment rate of 98%. The crowd estimation reached a high level of precision w.r.t. the ground truth, as demonstrated by the high accuracy obtained. A small error persists because Equation 4.4 uses a general probe request rate which includes all phases of devices. The rate is computed through simulation in which the devices change phases (and consequently, also rates) following the probabilistic distribution. In other traces, devices may act differently, due to a more varied use, in particular in real-case scenarios.

Figure 5.5 presents the specific outcomes derived from the data collected in room 14 at Politecnico di Torino, on the first day of the 2023/2024 academic year. In this context, the results produced by the framework combining clustering and time features have been selected for comparison with the detected ground truth because they exhibit the highest performance. The experiments involved conducting 10 two-minute captures during the transition between lessons to have a picture of the evolving trend of identified devices. This trend, both in terms of the ground truth and counting performed by the framework, is reported in the graph. The two lines illustrate the framework's ability to accurately estimate the number of devices in a given area and its capacity to track the changes in the number of individuals within the same environment.

The framework that combines clustering and time-related features has exhibited the most impressive performance. However, another aspect related to the capture time window requires attention. All the datasets used for tests reported in Table 5.4 have a time window inside which the present devices can send an acceptable number of probe requests. In real use case scenarios there can be the necessity to perform small captures, so that the parameter T in Equation 4.4, representing the time window of the capture, becomes very small. This problem could potentially narrow

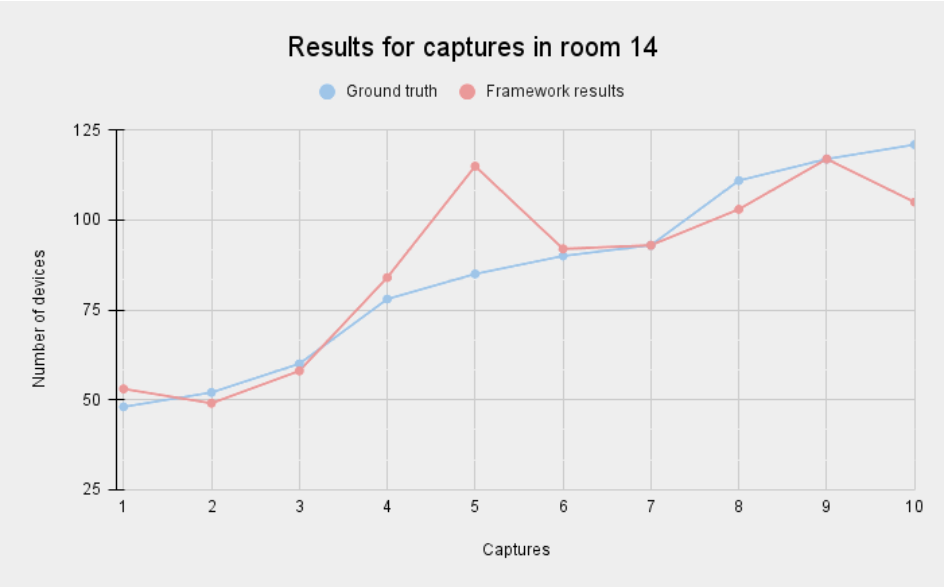

Figure 5.5. Comparison between ground truth and results obtained from the datasets collected in room 14 at Politecnico di Torino, on the first day of the 2023/2024 academic year.

the flexibility of the framework. These considerations brought to another set of tests done with specially designed datasets.

Particularly, each simulation:

- Contains the same number of devices, set to 5.
- Has devices of different models.
- Has a different capture time w.r.t. the others.

In environments with a lot of devices and a small time window, there is the risk of the sniffer not capturing any packet for certain devices. This happens because the number of probe requests sent is too high for the sniffing time set. The selection of a smaller ground truth was made to ensure that, in the shortest simulations, the sniffer could capture a sufficient number of packets (at least one) for each device. The comparisons between the ground truth and the outcomes for different time windows should give information about the smallest capture time that makes effective this solution.

Image 5.6 reports a graph presenting the results of the tests for the smallest time window search.

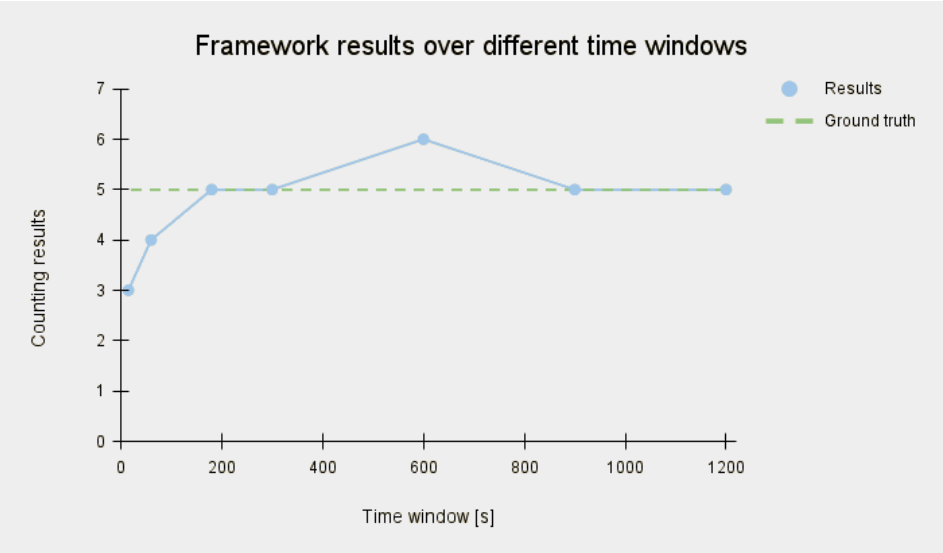

Figure 5.6. Relation between the outcomes and the time windows used.

The various time windows (i.e., the different simulation times that generated the datasets) are represented on the abscissa axis, while the count outcomes are found on the ordinate axis. The dotted line represents the ground truth.

Even if the time window gradually decreases, the results of the framework still maintain a high precision grade. With the shortest capture of just 15 seconds, the accuracy calculated with Formula 5.5 is still 60%. This result may appear not too satisfying compared with the accuracies of 80% and 100% obtained with bigger time windows. Again, in this case, the environments of the tests are all low-crowded, so a not-too-high accuracy is not a synonym for bad performance. With these considerations, the outcome retrieved with a time window of 15 seconds can be considered very satisfying. With shorter simulations, not all 5 devices were able to send at least one probe request, so it was decided to set 15 seconds as the inferior limit of the possible time windows.

These experiments have demonstrated that the framework that combines clustering techniques with time-related features proves to be effective across various use-case scenarios and with varying capture times.

Chapter 6

Future works

The domain of monitoring individuals within a particular area continues to have numerous challenges yet to be resolved. The necessity of various systems and frameworks over the years has underlined the profound significance of this research domain. Following the results achieved by this master's thesis, more extensive works can be carried forward. A primary initial stride would be enhancing the probe request generator's scope and capabilities.

A first release was published on GitHub [10] and it can be a good starting point to improve the quality of the generated traces. The project was released open source under MIT license [17] so that the research community can participate in future expansions. By updating the generator with new versions, the environments simulated would always be closer to real cases, both in terms of different situations and the behavior of devices.

The developed frameworks for crowd-monitoring were tested with synthetic datasets and real traces. More extensive work can be done by deploying the entire chain, starting from the data acquisition, to the final estimation. With the implementation of a working system that contains all the steps to elaborate input messages and supply the counting result, other analyses can be done. An additional assessment between the two frameworks could be conducted, along with an evaluation of the efficiency of clustering in combination with frequency features using various time windows. In general, the main future work should have the goal of bringing the developed systems into real scenarios to test their accuracy and performance.

Chapter 7

Conclusion

The proliferation of IoT and smart devices in today's world enables tracking and estimating the number of people in specific areas, making crowd-monitoring crucial for regulating various activities. This transformation allows for better quality of life through improved population movement management, including safety enhancements at events, managing pedestrian traffic, optimizing energy usage, and understanding customer behavior for marketing.

Chapter 1 provides a brief introduction to the main goals of the thesis, which involve examining the movement of individuals and exploring potential alternative approaches. In this particular scenario, the central focus is around the WiFi probe request signals emitted by smart devices owned by pedestrians passing by. Despite the other possible methods treated in Chapter 2, these specific types of messages usually exchanged through computer networks, are crucial for crowd-monitoring. An overview of the main features and problems related to probe requests is reported. This includes an in-depth analysis of the MAC address structure and its randomization adopted by vendors to increase people's privacy. The analysis delves into obsolete crowd-monitoring techniques involving probe requests and shows the reason why they are no longer used.

The first contribution brought by this thesis work is the development of a probe request traces generator, described in Chapter 3. The system has been modeled as a state machine where all possible environmental events are handled to build realistic traces. A set of models has been analyzed to extract the features and behaviors of their messages. This approach enables the generator to faithfully replicate the chosen devices with precision. With the development of this system, the lack of ground truth is not a problem anymore for the crowd-monitoring frameworks. Machine learning models could have a wide range of synthetic datasets to use for the training. Also, the validation becomes easier for the implemented systems, since the precise number of devices can be accurately managed within the generator settings.

Chapter 4 reports the second contribution of this thesis work, the development of two frameworks for crowd-monitoring. The first method involved an encoder-decoder neural network, a particular kind of deep learning architecture. The aim is to find a representation of probe requests that facilitates the division of these into device-oriented groups. The second system employs traditional clustering algorithms coupled with a heuristic derived from domain knowledge. The message groups are created by exploiting model-oriented features rather than device-oriented ones. After that, to derive the number of devices present for each model (i.e., for each cluster), the framework uses information about the frequency behavior of probe requests.

Both contributions have been evaluated with a series of tests, whose results are shown and explained in Chapter 5.

The probe request generator displayed a significant capability to emulate real device behavior. To validate this, we compared the generated traces with real ones using a set of collected metrics. The outcomes indicate a high degree of similarity between the synthetic and real data.

The testing phase on the two crowd-monitoring frameworks aimed to measure their accuracy in retrieving a precise counting result, comparable with the actual ground truth. Despite some challenges with the first method's results, the second system consistently demonstrated remarkable versatility and accuracy across all the datasets it was applied to.

Bibliography

[1] Bluetooth special interest group. bluetooth specification version 1.1 and 1.2. [2001]. [Online]. URL: http://www.bluetooth.com.

[2] Tomas Bravenec, Joaquín Torres-Sospedra, Michael Gould, and Tomas Fryza. Exploration of user privacy in 802.11 probe requests with mac address randomization using temporal pattern analysis, 2022.

[3] Ziqing Chen, Wei Yuan, Ming Yang, Chunxiang Wang, and Bing Wang. SVM based people counting method in the corridor scene using a single-layer laser scanner. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 2632–2637, 2016.

[4] Cross-entropy. [Online]. URL: https://en.wikipedia.org/wiki/Cross-entropy#cite_ref-2.

[5] Laihui Ding, Shengke Wang, Rui Li, Long Chen, and Junyu Dong. PC-PINet: Partial re-identification network for people counting with overlapping cameras. In *International Conference on Image, Vision and Computing (ICIVC)*, pages 66–71, 2021.

[6] Fatih Erden, Ali Alkar, and Ahmet Cetin. A robust system for counting people using an infrared sensor and a camera. *Infrared Physics & Technology*, 72, 08 2015.

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, KDD'96, page 226–231. AAAI Press, 1996.

[8] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council.

[9] Kalkidan Gebru, Marco Rapelli, Riccardo Rusca, Claudio Casetti, Carla Fabiana Chiasserini, and Paolo Giaccone. Edge-based passive crowd monitoring through WiFi beacons. *Computer Communications*, 192:163–170, 2022.

[10] Github link for probe request generator. [Online]. URL: https://github.com/riccardo-rusca/ProbeRequestGenerator.

[11] Andrei Günter, Stephan Böker, Matthias König, and Martin Hoffmann. Privacy-preserving people detection enabled by solid state LiDAR. In *International Conference on Intelligent Environments (IE)*, pages 1–4, 2020.

[12] Json official website. [Online]. URL: https://www.json.org/json-en.html.

[13] Library for adam optimization algorithm. [Online]. URL: https://pytorch.org/docs/stable/generated/torch.optim.Adam.html.

[14] Lidar sensor schema. [Online]. URL: https://www.elprocus.com/lidar-light-detection-and-ranging-working-application/.

[15] John D.C. Little. A proof for the queuing formula: L = λw. *Operations Research*, 2(4):447–457, 1954.

[16] Jeremy Martin, Travis Mayberry, Collin Donahue, Lucas Foppe, Lamont Brown, Chadwick Riggins, Erik Rye, and Dane Brown. A study of MAC address randomization in mobile devices and when it fails. *Proceedings on Privacy Enhancing Technologies*, 2017.

[17] Mit license terms and conditions. [Online]. URL: https://en.wikipedia.org/wiki/MIT_License.

[18] MS Windows NT kernel description. [Online]. URL: https://gs.statcounter.com/vendor-market-share/mobile/europe/yearly-2020-2023-bar.

[19] Ei Phyu Myint and Myint Myint Sein. People detecting and counting system. In *IEEE Global Conference on Life Sciences and Technologies (LifeTech)*, pages 289–290, 2021.

[20] Michele Nitti, Francesca Pinna, Lucia Pintor, Virginia Pilloni, and Benedetto Barabino. iabacus: A wi-fi-based automatic bus passenger counting system. *Energies*, 13(6):1446, 2020.

[21] Numpy library. [Online]. URL: https://numpy.org/.

[22] Cristian Perra, Amit Kumar, Michele Losito, Paolo Pirino, Milad Moradpour, and Gianluca Gatto. Monitoring indoor people presence in buildings using low-cost infrared sensor array in doorways. *Sensors*, 21(12), 2021.

[23] Lucia Pintor and Luigi Atzori. A dataset of labelled device wi-fi probe requests for mac address de-randomization. *Computer Networks*, 205:108783, 2022.

[24] Pir sensor schema. [Online]. URL: https://components101.com/sensors/hc-sr501-pir-sensor.

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

[26] Pyshark library. [Online]. URL: https://pypi.org/project/pyshark/.

[27] Python programming language. [Online]. URL: https://www.python.org/.

[28] Random library for python programming language. [Online]. URL: https://docs.python.org/3/library/random.html.

[29] Raspberry pi foundation. [Online]. URL: https://www.raspberrypi.com/.

[30] Riccardo Rusca, Filippo Sansoldo, Claudio Casetti, and Paolo Giaccone. What WiFi probe requests can tell you. In *IEEE Consumer Communications & Networking Conference (CCNC)*, pages 1086–1091, 2023.

[31] Scapy library. [Online]. URL: https://scapy.net/.

[32] Aleš Simončič, Miha Mohorčič, Mihael Mohorčič, and Andrej Hrovat. Non-intrusive privacy-preserving approach for presence monitoring based on wifi

probe requests. *Sensors*, 23(5), 2023.

[33] Sklearn library. [Online]. URL: `https://scikit-learn.org/stable/`.

[34] Marco Uras, Enrico Ferrara, Raimondo Cossu, Antonio Liotta, and Luigi Atzori. Mac address de-randomization for wifi device counting: Combining temporal- and content-based fingerprints. *Computer Networks*, 218:109393, 2022.

[35] Wireshark software suite's official website. [Online]. URL: `https://www.wireshark.org/`.