



**Politecnico
di Torino**

POLITECNICO DI TORINO

Master Degree course in Communications and Computer Networks
Engineering

Master Degree Thesis

Privacy-Preserving People Flow Monitoring with Bloom Filters

Supervisors

Prof. Claudio Ettore CASETTI

Prof. Paolo GIACCONE

PhD. Riccardo RUSCA

Candidato

Alex CARLUCCIO

ACADEMIC YEAR 2022-2023

Acknowledgements

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

I want to express my deep gratitude to Riccardo Rusca and Diego Gasco for their valuable contribution and support during the completion of this thesis. Their dedication and collaboration greatly enriched this work, and I appreciate the energy and commitment they devoted to the project.

Furthermore, special thanks go to my family, my girlfriend Marta, and all my friends for their constant support and the motivation they provided me with over these years. Your affection and encouragement have been crucial to my journey, and I am grateful to have such extraordinary people in my life.

Abstract

In the last decade, our cities have undergone significant transformation due to the widespread adoption of IoT (Internet of Things) sensors and smart devices owned by citizens, resulting in significant improvements in urban life.

One of the most common and relevant research fields concerns the monitoring of people’s movements, with the aim of enhancing services for citizens. This type of monitoring also pertains to optimizing transportation infrastructure, bike lanes, and public thoroughfares. However, it is also of crucial importance in identifying and quantifying individuals in sensitive areas, such as ensuring safety during large-scale events or at transit points. Additionally, it is essential to introduce new technologies for tracking individuals in emergency situations, to reduce response times and increase the likelihood of success.

In the context of this thesis, we focus on the WiFi fingerprinting technique, which uses the MAC address of mobile devices as a proxy for people counting. Due to European GDPR regulations and the stringent measures taken by major smart device providers to enhance user privacy (e.g., through MAC address randomization), most of the techniques examined in the past must be rethought and redefined.

For this reason, we have adopted efficient probabilistic data structures called Bloom filters for storing sensitive information. Thanks to their formal ”deniability” property, we are able to offer a solution that safeguards user privacy. Our solution is also compatible with trajectory-based crowd monitoring, as these elements enable set operations, such as intersection, for determining flows.

Contents

List of Figures	1
List of Tables	3
1 Introduction	5
2 Technological issues	7
2.1 WiFi probe request	7
2.2 Bluetooth probe request	10
2.3 MAC address	11
2.3.1 MAC address randomization	11
2.3.2 How main OS implement MAC address randomization	12
2.4 Privacy	13
2.4.1 GDPR	14
2.4.2 Anonymization Techniques	15
2.4.3 Bloom filter	17
2.5 Data Transport	18
2.5.1 MQTT	19
2.5.2 CoAP	21
3 Proposal architecture	25
3.1 WiFi scanner	26
3.2 Chain of processing	27
3.2.1 De-randomizer	27
3.2.2 Bloom filter	28
3.2.3 Data Transfer	29
4 Bloom filter for flow tracking	31
4.1 Tuning	31
4.2 Privacy	32
4.2.1 Brute-Force Attack	32
4.3 Privacy Protection	34

4.3.1	Anonymization Noise	35
4.3.2	Multiple Anonimity	36
4.4	Counting	38
4.4.1	Intersection	38
5	Implemented Solution	41
5.1	Hardware	41
5.2	System Description	43
5.2.1	Remote Accessing to Raspberry	43
5.2.2	Capturing WiFi Probe Request	44
5.2.3	Bloom Filter Dimension	44
5.3	Python Chain Of Processing	44
6	Experimental evaluation	49
6.1	Profiling	49
6.2	Bloom filter dimension	55
6.3	Counting	57
6.3.1	Counting intersection	58
6.4	Anonimity	60
6.4.1	Multiple Anonimity	61
7	Conclusion and Future Work	63
	Bibliography	65

List of Figures

2.1	Simplified probe transmission scheme	8
2.2	Structure of probe request frame (reproduced from [1])	8
2.3	Communication channels used by WiFi networks (reproduced from [1])	9
2.4	Inquiry process	10
2.5	48-bit MAC Address Structure	12
2.6	Hash function	15
2.7	Salted hash function	16
2.8	Truncated hash function	17
2.9	Bloom filter	17
2.10	QoS 0	20
2.11	QoS 1	20
2.12	QoS 2	21
2.13	Structure of CoAP message (reproduced from [12])	22
3.1	Proposal architecture	25
3.2	HT Capabilities	26
4.1	Hiding set	35
4.2	Bloom filter with $\gamma = 1$ composed of elements (x_1, x_2, x_3)	37
4.3	Bloom filter with $\gamma = 0.66$ composed of elements (x_1, x_2, x_3)	37
4.4	Bloom filter with $\gamma = 1$ and $K = 3$ composed of elements (x_1, x_2, x_3)	38
5.1	Raspberry Pi 3 Model B	42
5.2	Raspberry Pi 3 Model B Configuration	43
5.3	System Python Chain of Processing	45
5.4	System Flow Chart	47
6.1	Middleware Script CPU Load Evolution Over Time	50
6.2	Middleware Script RAM Load Evolution Over Time	50
6.3	Analyzer Script CPU Load Evolution Over Time	51
6.4	Analyzer Script RAM Load Evolution Over Time	51
6.5	Filter Script CPU Load Evolution Over Time	52
6.6	Filter Script RAM Load Evolution Over Time	52

6.7 Derandomizer Script CPU Load Evolution Over Time	53
6.8 Derandomizer Script RAM Load Evolution Over Time	53
6.9 Bloomfilter Script CPU Load Evolution Over Time	54
6.10 Bloomfilter Script RAM Load Evolution Over Time	54
6.11 Evolution of false positive probability during the insertion of elements into a Bloom filter	55
6.12 Evolution of false positive probability during the insertion of elements into different Bloom filters. Each filters is configured with $m = 10000$	56
6.13 Evolution of false positive probability with different values of k	56
6.14 Comparison between the number of elements actually inserted and the counted ones.	57
6.15 Relative error comparison between the number of elements actually inserted and the counted ones.	58
6.16 Comparison between the estimators c_1 and c_2 to evaluate the number of people moving from one scanner to the other	59
6.17 Relative error of the estimators c_1 and c_2 to evaluate the number of people moving from one scanner to the other	60
6.18 γ -deniability value in relation to the number of inserted MAC addresses	61
6.19 γ -deniability value, for different K , in relation to the number of inserted MAC addresses	62

List of Tables

2.1 Notations and definitions	18
-----------------------------------------	----

Chapter 1

Introduction

Over the past years, Information and Communication Technologies (ICT) have experienced continuous development, thanks to advancements in both hardware and software. The implementation of these technologies in cities has led to a significant improvement in urban operations, with various terms coined to describe these realities, such as "cyberville", "digital city" and "smart city".

The concept of a "smart city" represents the broadest abstraction among all the used definitions, but there is still no final and universally accepted definition by the academic community and industry experts. In simple terms, a smart city is a place where traditional networks and services are made more efficient, flexible, and sustainable through the use of information, digital, and telecommunication technologies. The goal is to improve the city's functioning for the benefit of its inhabitants. In other words, a smart city leverages digital technologies to provide high-quality public services to its citizens while optimizing resource usage and reducing environmental impact.

Monitoring people flows is another significant focus of digital city that aims to enhance citizen services and ensure safety in sensitive areas and during transit. It involves using advanced tracking technologies, which are particularly important during health emergencies like the ongoing situation.

The world's population has significantly increased in recent decades, as have expectations regarding living standards. Projections indicate that by 2050, approximately 70% of the world's population will reside in urban areas.

Turin, being the seventieth city worldwide and the fifth in Italy for the number of in-person events held in 2022, faces the challenge of managing a high flow of constantly moving people. This situation makes accurate people counting and crowd monitoring essential elements for making effective decisions to ensure public safety. The ability to precisely assess crowd dynamics, estimate necessary resources, and optimize emergency response efforts is of vital importance for the authorities. However, the task of counting and tracking individuals during large gatherings or chaotic situations has always been a complex challenge.

The aim of this work is to find a solution to this problem using scanner (APs) for scanning WiFi bands and probabilistic data structures called Bloom filters for storing data classified as personal while ensuring complete privacy.

During this study, various issues and limiting factors will be addressed, some of which are due to the system itself, including:

- Ability to detect only users carrying smart devices with enabled WiFi interfaces.
- Ability to detect a flow of users only under specific MAC address randomization conditions.
- Inability to distinguish if a user is carrying multiple devices simultaneously, such as smartwatches, smartphones, and tablets.
- Inability to fully counter the randomization of MAC addresses.

The second and last points in the list require important clarification. In recent years, mobile device manufacturers have introduced MAC address randomization techniques. This means that the same device uses random and different MAC addresses in various generated packets. This must be carefully managed by choosing the most effective de-randomization systems to avoid counting the same client multiple times. It is also evident that monitoring flows under these circumstances is extremely complicated.

Alternative solutions exist, such as those proposed by companies like Footfall Cam, which offers various models. FootfallCam 3D Prowave uses depth data and a custom radar system to measure area occupancy for collaboration. However, its range of action is very limited, making the study of flows impractical. On the other hand, the FootfallCam 3D Pro2 utilizes three-dimensional stereoscopic vision technology to capture images, and its GPU runs artificial intelligence algorithms that leverage depth, color, and texture models to identify individuals. However, in this case, the main issue is privacy as it involves working with images of monitored individuals.

Chapter 2

Technological issues

In this chapter, we will discuss the current state of WiFi probe requests, the implementation of MAC address randomization in our smart devices, and touch upon Bloom filters and their properties.

2.1 WiFi probe request

Discovering networks by scanning all possible channels and listening to beacons is not considered very efficient (passive scanning). To improve this discovery process, devices often use what is called active scanning.

In active scanning, devices iterate through all available channels and instead of passively listening to signals in search of a network, they send a frame called Probe request, asking which networks are available on that channel.

The Probe requests are sent to the broadcast address ($FF:FF:FF:FF:FF:FF$). Upon receiving a probe request, an Access Point (AP) has the option to replay the client of its presence by sending a probe response. However, if there are no APs nearby or if the APs are in passive mode, the client will not receive any response.

Once a request is sent, the device starts a countdown called Probe Timer while waiting for responses. At the end of the timer, the device processes the received responses, and if nothing is received, it moves to the next channel and repeats the discovery process. The process is illustrated in 2.1.

Devices sending Probe requests can specify the SSID they are looking for if they are searching for a particular network. In such cases, only IBSS (*Independent Basic Service Set*) stations or access points (APs) that support that SSID will respond. The value of the SSID (*Service Set Identifier*) can also be set to 0 (i.e., the SSID field is present but empty). This is called a "Wildcard" SSID or "Null" Probe request.

In 2.2, there is an example of the structure of a Probe request. We can now analyze a list of information that can be found within it:

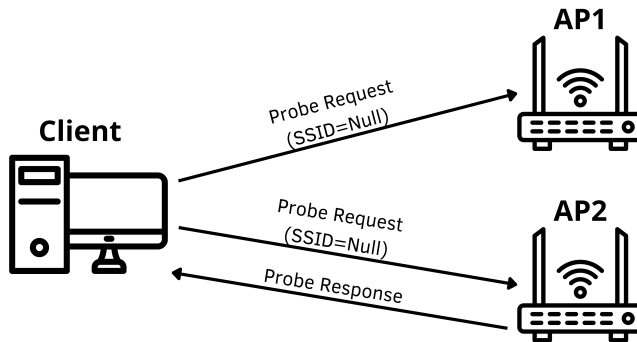


Figure 2.1. Simplified probe transmission scheme

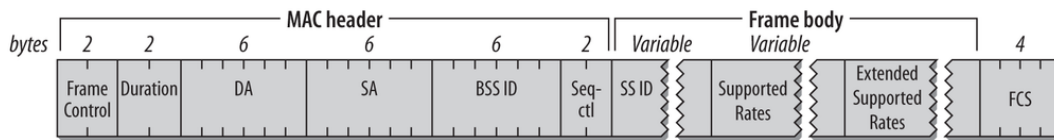


Figure 2.2. Structure of probe request frame (reproduced from [1])

- SSID: The name by which a WiFi network or WLAN identifies itself to its users.
- Supported Rates: Data rates supported by the device to enable a correct connection; all indicated rates must be supported.
- Request Information: Used to communicate various capabilities, preferences, and requirements of the client device to the access points
- Extended Supported Rates: Mandatory when the number of supported rates exceeds eight; otherwise, its presence is optional.
- DSSS Parameter Set: Used to indicate the channel.
- Supported Operating Classes: List of operating classes or frequency ranges that the client device is capable of using. Each operating class corresponds to a specific set of WiFi channels.
- HT Capabilities: MCS (Modulation and Coding Scheme) values which are supported by the wireless network.
- 20/40 BSS Coexistence: Feature that allows devices to operate in both 20 MHz and 40 MHz channel bandwidths in the same Basic Service Set (BSS).

- **Extended Capabilities:** Used to indicate specific features or functionalities that the device supports beyond the basic capabilities defined in the standard Wi-Fi specifications (802.11k, 802.11v, 802.11r)
- **SSID List:** List of Service Set Identifiers (SSIDs) that the WiFi client device is seeking or has previously connected to.
- **Channel Usage:** Information about the current channel utilization of a WiFi client device.
- **Interworking:** Indicate the client device’s support for specific interworking features. Interworking is the ability of a WiFi client to seamlessly connect and switch between different types of networks, such as WiFi, cellular networks, and other wireless technologies, in a unified and efficient manner.
- **Mesh ID:** Used by device in a Mesh network to identify itself and indicate its presence as a potential node or participant.
- **Vendor Specific:** Used from manufacturers to include custom or proprietary information in the probe request, which can be utilized by specific vendor-specific features, services, or protocols.

Scans are essential to maintain WiFi connections during the device’s sleep state, constantly seeking better signal strength to improve the user experience. Furthermore, search requests are crucial for connecting to hidden networks where access points (AP) do not broadcast beacons.

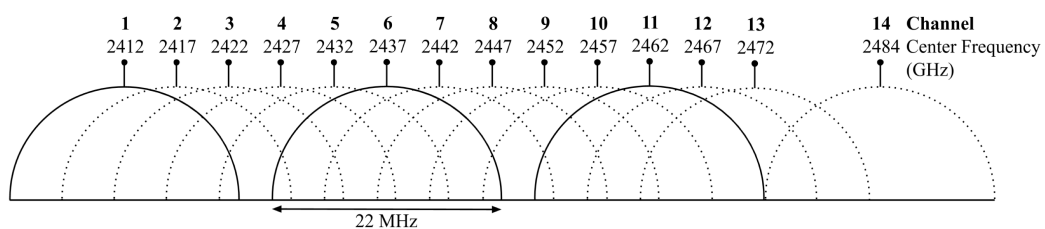


Figure 2.3. Communication channels used by WiFi networks (reproduced from [1])

In 2.3, there is a depiction of the 14 distinct channels utilized by WiFi networks, along with the central frequency spectrum for each of these channels. The illustration emphasizes Channels 1, 6, and 11, which are commonly employed by wireless routers due to their lack of frequency spectrum overlap between one another.

2.2 Bluetooth probe request

Bluetooth devices function within the globally unlicensed 2.4 GHz short-range radio frequency spectrum. This unlicensed frequency allows Bluetooth technology to be used without the need for regulatory approval or licensing, enabling its widespread adoption worldwide.

When a device has its Bluetooth interface activated, it can be in two distinct states, each serving different purposes and functions:

- *Discoverable State*: In this mode, the Bluetooth device actively broadcasts its presence to other nearby Bluetooth devices, making itself visible and available for pairing and communication. When in this state, the device is actively seeking to establish connections with other compatible devices.
- *Non-Discoverable State*: When a device is in the non-discoverable state, it does not actively broadcast its presence to other devices in the proximity. This mode is useful for enhancing privacy and security, as the device remains hidden and will not be detected by other Bluetooth devices during normal scanning procedures. However, even in this state, the device can still connect to other devices that are in the discoverable state, as long as they initiate the pairing process.

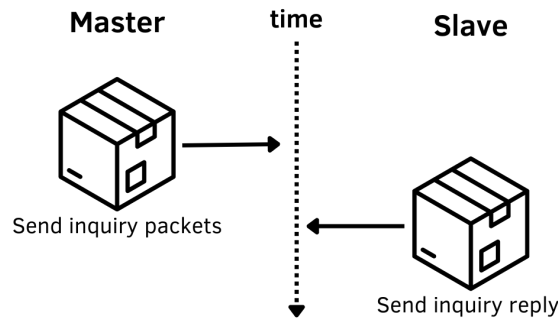


Figure 2.4. Inquiry process

Bluetooth's ability to switch between these two states allows for efficient and controlled communication, striking a balance between user convenience and privacy. Users can decide when and how their devices interact with others, ensuring a seamless and secure Bluetooth experience.

Bluetooth networks utilize a master/slave model to regulate the transmission of data, determining when and where devices can exchange information. To discover other Bluetooth devices, the Bluetooth protocol employs a method similar to WiFi probe requests, known as the "inquiry mode", as showed in 2.4. In this mode, a

Bluetooth device sends out inquiry packets, essentially broadcasting a request to find nearby devices. Other devices within range and in inquiry scan mode can respond with inquiry replies, providing the necessary information for establishing a connection.

During the inquiry process, devices exchange relevant data, such as their MAC addresses and other identification information, facilitating the setup of a connection between them.

2.3 MAC address

MAC address stands for Media Access Control Address, which serves as a unique identifier for an IEEE 802 network interface. IEEE 802 encompasses various standards used in networking technologies, including Ethernet, WiFi, ZigBee, FDDI (Fiber Distributed Data Interface), and Bluetooth, among others. Each network interface is assigned a distinct MAC address, allowing devices to be uniquely identified.

A device equipped with both a WiFi card and a Bluetooth card will have two MAC addresses, one for each card. Specifically, a MAC address is a 48-bit code created to be unique. It consists of two parts, each composed of 3 bytes: the Organizationally Unique Identifier (OUI), which represents the manufacturer of the network interface, and the serial number of the card itself. Additionally, the IEEE offers the option to purchase a "private" OUI, which does not include the company's name in the registry.

MAC addresses are divided into two categories:

- **Universally Administered Addresses:** These addresses are assigned to devices by manufacturers and are sometimes referred to as "burned-in addresses." In this case, the requirement is to be globally unique.
- **Locally Administered Addresses:** These addresses are assigned by the network administrator, replacing the burned-in address. Locally administered addresses do not include the OUI and are not required to be globally unique.

The distinction between universally administered addresses and locally administered addresses is determined by the second least significant bit of the first byte (as shown in 2.5). If the bit is set to 0, the address is considered universally administered; if it is set to 1, the address is considered locally administered.

2.3.1 MAC address randomization

Due to the fact that the MAC address serves as a unique identifier for a device, it can be utilized to identify an individual. As explained in 2.2 and 2.1, both WiFi devices' probe requests and Bluetooth devices' inquiry packets require a MAC address.

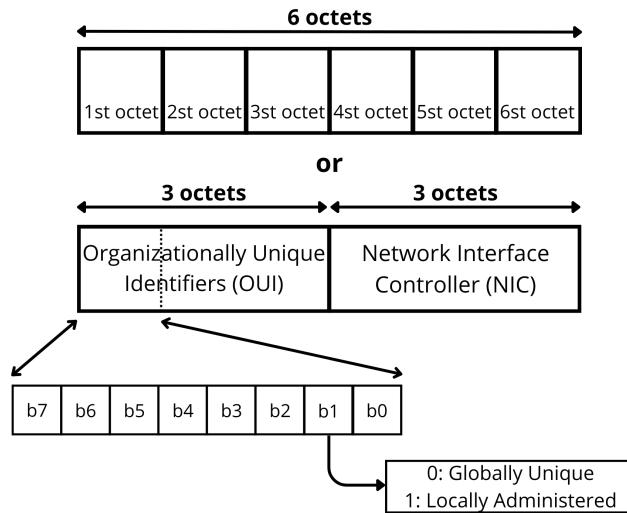


Figure 2.5. 48-bit MAC Address Structure

Consequently, smart devices belonging to users can be effortlessly tracked across different time periods and locations as they continuously transmit their distinctive identity.

To address this issue, a mechanism for randomizing MAC addresses has been developed. When a device attempts to connect to an access point (AP), it sends packets containing different source MAC addresses, preventing tracking attempts. Due to this, in recent years, several public WiFi networks have been created in various locations to easily monitor the flow of people. It should also be noted that with recent updates to various operating systems, MAC address randomization is not only used during the search phase but also during the connection phase to the network.

2.3.2 How main OS implement MAC address randomization

In the following we will discuss how major operating systems implement MAC address randomization. It is crucial to note that randomization techniques and their application on smart devices are constantly evolving. With each new software version, manufacturers frequently make adjustments to improve user privacy and enhance the effectiveness of these techniques.

Android

Starting from Android 8.0, as reported in [2], Android devices use randomized MAC addresses when searching for new networks while not currently connected to a network.

In Android 9, there is also an option in the developer settings (disabled by default) that allows the device to use a randomized MAC address during the connection to a Wi-Fi network.

In Android 10, MAC address randomization is enabled by default for client mode, SoftAp, and Wi-Fi Direct.

Apple OS

Apple platforms, as mentioned in [3], use random MAC addresses both during scans for WiFi networks and during ePNO (enhanced Preferred Network Offload) scans, both when the device is not connected to a WiFi network and when the processor is in a low-power state. ePNO scans are used to provide location-based services on the device, such as automations triggered when arriving at a location with a specific WiFi network. From iOS 14, watchOS 7, iPadOS 14, and onwards, whenever a device connects to a WiFi network, it identifies itself using a random MAC address. Users have the possibility to disable this feature, either manually or through a new setting in the WiFi configuration. Under certain circumstances, the device will revert to using its actual MAC address.

Windows

Windows systems, as stated in [5], do not use randomized MAC addresses by default, but depending on the versions of the operating system, it is possible to enable this feature. Users can choose between using a random MAC address for all WiFi networks or only for a specific WiFi network to which they are connected.

2.4 Privacy

The idea of privacy has captured considerable attention from both scholarly researchers and the general media in recent times. Privacy lacks a single, unambiguous, and straightforward definition. Its interpretation hinges on the specific circumstances in which it is employed. For instance, during the period when non-electronic newspapers, magazines, and photographs were prevalent, privacy was conceived as the "right to solitude". When journalists compose articles about individuals' experiences or lives in newspapers and release pictures of them without their consent, this act could be considered an infringement on privacy since these individuals possess the right to seclusion and should not have their images distributed publicly, particularly without their agreement.

Nevertheless, new modes of communication have surfaced due to technological advancements, furnishing individuals with more opportunities to explore and exchange information. Consequently, the notion of privacy carries a somewhat distinct implication in this situation, as users generally have the option to decide whether or not to disclose information about themselves. To safeguard individuals' privacy in this new era, new regulations have been introduced by competent authorities worldwide; for example, the GDPR issued by the European Union through the European Data Protection Board.

2.4.1 GDPR

The GDPR (General Data Protection Regulation), also known as RGD in Italian, is a new European regulation concerning the protection of personal data [11]. Its goal is to harmonize rules regarding the collection and processing of data. Data under GDPR can be divided into two different groups:

- Sensitive data: revealing information about race, religious, political, sexual orientation, health status, economic and social status.
- Identifying data: personal details, residential address, and images allowing direct identification of an individual. The GDPR also protects online identifiers, such as email addresses, cookies, IP addresses, and geolocation data.

The new Privacy Regulation applies to both individuals and legal entities (professionals and companies) processing personal data of European citizens, whether inside or outside the European Union, online or offline. The GDPR involves four key actors in data protection:

- Data Subject refers to an individual who possesses the data provided and is entitled to the rights established by the new Regulation, excluding legal entities such as companies.
- Data Controller refers to companies to whom users willingly share their data (with prior consent to privacy). This role involves deciding the objectives and procedures for data processing while implementing preventive measures to ensure compliance with privacy protection regulations, including privacy by design. Legally, they have a responsibility to meet the requirements outlined by the regulation.
- Data Processor refers to a person appointed by the Data Controller to jointly implement technical and organizational measures to ensure data security.
- Data Protection Officer (DPO or RPD) refers to a person that is responsible for ensuring adherence to the Regulation. They must possess specialized knowledge of data protection regulations. Their appointment is compulsory in cases explicitly stipulated by the GDPR.

2.4.2 Anonymization Techniques

Anonymization techniques are strategies and processes used to remove or alter identifying information from personal data. They enable data to be rendered anonymous or pseudonymous, thereby reducing the risk of recognition. The main objective is to balance the use of data for legitimate purposes such as research and analysis, with the protection of individuals' rights and privacy.

Hash

Hash functions are cryptographic algorithms used to convert data of any size into a fixed-length value, known as a "hash" or "digest". These algorithms are designed to be one-way, meaning it should not be possible to reverse the function and go from the digest back to the plain text (the conditional is used because certainty cannot be guaranteed). As shown in 2.6, we can see an example of how it works. Each input will be divided into fixed-sized blocks, which are called data blocks (the size depends on the algorithm). If the blocks are not large enough, padding may be added to fill them out. The hash function is then iterated as many times as the number of data blocks, and each time, the output of the previously data block is used as the input for the next block. Then the final output can be represented in hexadecimal form.

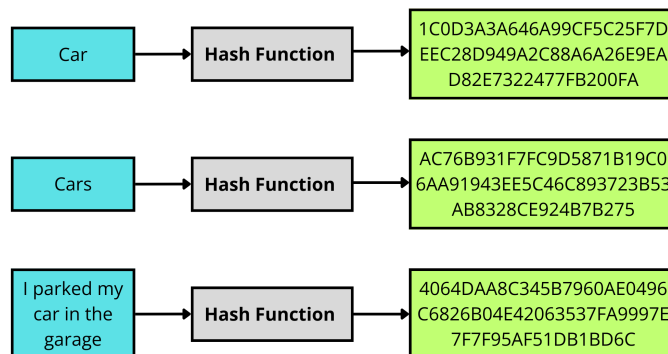


Figure 2.6. Hash function

Hash functions have several important characteristics:

- **Uniqueness:** Every input will always produce the same output when the same hash function is used. However, two different inputs should produce completely different outputs.
- **Speed:** Hash functions must be efficient and fast in calculating the digest to support the processing of large amounts of data.

- Collision resistance: A collision occurs when two different inputs produce the same output. Hash functions are designed to minimize the probability of this happening as much as possible, making it difficult or practically impossible to find two different values with the same hash.
- Data sensitivity: Even a small variation in the input data should produce a completely different hash. This property is known as "diffusion" and contributes to the security of the hash function.

Salted Hash

Salted hashing is a technique used to enhance security by introducing additional layers of randomness during the hashing process as shown in 2.7. A random string called "salt" is added to the input before hashing, making it challenging for an attacker to determine the original plaintext without access to both the salt and the hashed value. The concept of salting provides several key benefits for data security. Firstly, it mitigates the vulnerability of common attacks such as rainbow table attacks, wherein precomputed tables of hashed values are used to expedite the reversal of hashes. Furthermore, salting offers defense against brute-force attacks that attempt to guess the plaintext through an exhaustive search of possible inputs.





				
Password	P4ssW0rD	P4ssW0rD	P4ssW0rD	P4ssW0rD
Salt	-	-	ehts43	cd6els
Hash	FA90CE1	FA90CE1	DE25C1A	B78AC29

Figure 2.7. Salted hash function

Truncated Hash

Truncated hashing is a process in which the output of a hash function is shortened to a specific number of bits. Instead of producing the full hash value, only a portion of it is retained, as is possible to see in 2.8. Is also important to note that truncating a hash reduces the entropy and increases the risk of collisions, where different inputs produce the same truncated hash.

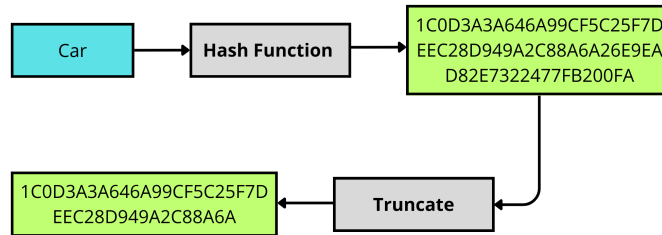


Figure 2.8. Truncated hash function

2.4.3 Bloom filter

Bloom filters are probabilistic data structure used to represent groups of elements. It is constructed using an array of bits $B[i]$ with a length m and k hash functions. Initially, the array's bits are set to 0.

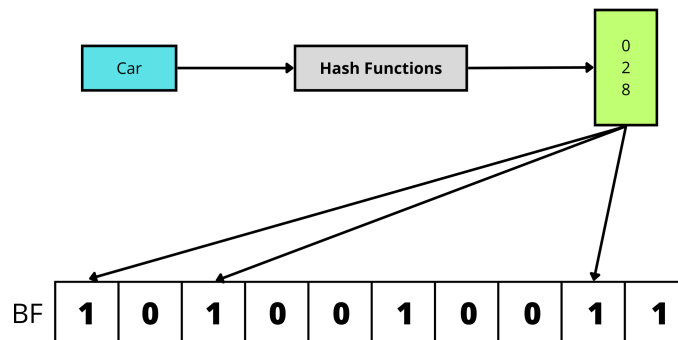


Figure 2.9. Bloom filter

When an element has to be inserted into the Bloom filter, the input is processed through the k hash functions, which determine the positions in the array where the value 1 should be inserted.

To check if an element is present in a Bloom filter, similar steps to insertion are performed. The element is passed through the k hash functions, which indicate which array cells to check. If all the indicated bits have a value of 1, the element is considered "probably present" in the Bloom filter. However, if even a single bit has a value of 0, the element is "definitely not inserted" in the Bloom filter. Before proceeding, in 2.1 is possible to have a reminder of the notation used.

Table 2.1. Notations and definitions

Notation	Definition
S	Set of element stored in the Bloom filter
$BF(S)$	Bloom filter storing a set S
$n = S $	Number of element stored in the Bloom filter
m	Size in bit of the Bloom filter
k	Number of hash functions
t_A	Number of bits set to 1 in the Bloom filter A
V	Hiding Set
U	Set of elements in the universe
$ U $	Number of elements in U

False Positive

By encoding elements from a large universe into a smaller universe represented by a bit-vector, there is a probability, calculable through appropriate equations, that an element x not belonging to S has collisions for each output of the k hash functions. When this happens, it is referred to as a *false positive*. Equation 2.1 below shows how to calculate this probability.

$$\Pr(\text{false positive}) = (1 - p)^k = \left(1 - \left(1 - \frac{1}{n}\right)^{mk}\right)^k \simeq \left(1 - e^{-\frac{km}{n}}\right)^k \quad (2.1)$$

To mitigate false positives, it is possible to adjust the parameters of the Bloom filter, such as the size of the array and the number of hash functions, according to the specific needs of the application. For this reason, given m and n , it is possible to calculate the optimal number of hash functions using the following formula:

$$k_{opt} = \frac{n}{m} \log(2) \quad (2.2)$$

2.5 Data Transport

Two major categories of application layer protocols exist in the realm of the Internet of Things (IoT): Client/Server and Publish/Subscribe. CoAP (Constrained Application Protocol) and MQTT (Message Queuing Telemetry Transport) stand as the two most prevalent protocols, with the former being associated with the Client/Server family and the latter falling under the Publish/Subscribe family.

2.5.1 MQTT

The Message Queuing Telemetry Transport (MQTT) protocol had its genesis in 1999 thanks to an idea by Andy Stanford-Clark (IBM) and Arlen Nipper (Eurotech). In MQTT workflow three primary "entities" play pivotal roles:

- The Publisher: This is the device responsible for transmitting data to a particular set of subscribers.
- The Subscriber: A Subscriber represents a device that seeks to obtain data related to specific topics of interest.
- The Broker: Positioned at the core of the architecture serves as central hub. Every client, whether functioning as a publisher or subscriber, establishes a connection with the broker for the purpose of either receiving or sending messages.

The channels through which messages are disseminated are termed as topics. A topic is an identified logical conduit denoted by a UTF-8 string segmented by the forward slash "/" symbol, which is recognized as the separator for topic levels. Each segment of the topic separated by "/" signifies a distinct topic level. Every message conveyed within MQTT pertains to a topic, and the broker employs topic-based filtration to dispatch the message to all subscribers demonstrating interest in that particular topic.

In MQTT, wildcards offer a robust mechanism to enable the simultaneous subscription to multiple topics. When a client opts to subscribe to a specific topic, it can choose to either subscribe to precisely that topic or leverage wildcards to expand the scope of its subscription. It's noteworthy that wildcards can solely be applied to subscriptions and are not applicable for the purpose of message publication. Two distinct categories of wildcards exist:

- Single level wildcard, denoted by the plus symbol (+), facilitates the substitution of a single topic level. When subscribing to a topic featuring a single-level wildcard, any topic including an unspecified string in place of the wildcard will be matched.
- Multi level wildcard encompasses numerous levels within a topic. It is represented by the hash symbol (#) and should be positioned as the final character of the topic, following a forward slash.

For example, if the following list of topics is present, using the expression [2.3](#) will subscribe to the first and third topics. Conversely, with the expression [2.4](#), you will subscribe to all the topics in the list.

```

/home/groundfloor/livingroom/temperature
/home/groundfloor/livingroom/humidity
/home/groundfloor/kitchen/temperature
/home/groundfloor/kitchen/humidity

```

$$/home/groundfloor/ + /temperature \quad (2.3)$$

$$/home/groundfloor/# \quad (2.4)$$

MQTT introduces three tiers of Quality of Service (QoS). The determination of QoS takes place in mutual agreement between the sender and receiver (publisher-broker and broker-subscriber), thereby rendering it unnecessary for the QoS level to be consistent from the publisher to the subscriber. These three tiers encompass:

- QoS 0 (Figure 2.10): Approach based on best efforts where the sender does not anticipate acknowledgment or assurance of message reception. Consequently, the recipient does not acknowledge the receipt of the message, and the sender does not retain or retransmit it. QoS 0 is often referred to as "fire and forget" wherein the message is dispatched without subsequent monitoring or validation.

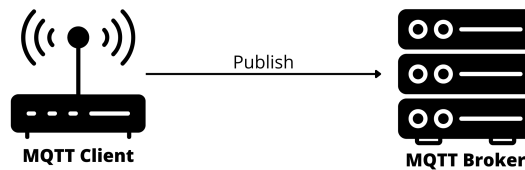


Figure 2.10. QoS 0

- QoS 1 (Figure 2.11): The primary objective is to guarantee message delivery to the receiver at least once. When a message is published the sender retains a duplicate of the message until it obtains a PUBACK packet from the recipient.

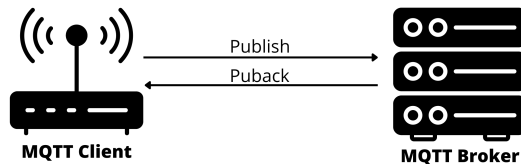


Figure 2.11. QoS 1

In the event that the sender does not receive the PUBACK packet within a reasonable timeframe, it initiates a retransmission of the message.

- QoS 2 (Figure 2.12): Represents the most elevated level of service guaranteeing precise single-time delivery of each message to the intended recipients. To achieve this, QoS 2 employs a four-step interaction. When the recipient receives a QoS 2 PUBLISH packet, it processes the publish message and responds by sending a PUBREC packet to acknowledge the receipt of the PUBLISH packet. In cases where the sender fails to receive a PUBREC packet from the recipient, it persists in transmitting appending a duplicate (DUP) flag. Following the successful reception of a PUBREC packet the sender responds with a PUBREL packet. Once the recipient obtains the PUBREL packet, it discards all stored states and responds with a PUBCOMP packet. Upon the culmination of the QoS 2 process, both parties attain assurance of the message’s delivery.

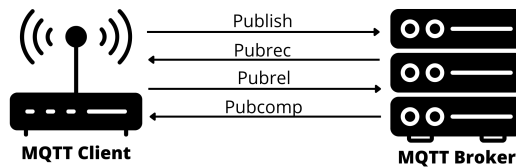


Figure 2.12. QoS 2

2.5.2 CoAP

CoAP stands for Constrained Application Protocol, and it is a protocol optimized for communication between devices with limited resources. It was developed by the IETF (Internet Engineering Task Force) group known as CoRE (Constrained Resource Environments). The interaction model is similar to that of HTTP (client-server), but it differs in the context of M2M (Machine-to-Machine) communication, where nodes can act as both clients and servers without a distinct differentiation of entity roles. CoAP packets are significantly smaller compared to typical TCP flows in HTTP, as CoAP utilizes UDP (User Datagram Protocol) for communication. Retransmissions and packet ordering are managed at the application level.

In CoAP, there are four types of messages defined:

- Confirmable: To ensure greater reliability, a message can be marked as "confirmable"; in such cases, the message requires a response (Acknowledgment). Assuming no packet losses, each confirmable message will be followed by an acknowledgment.

- Non-confirmable: Messages labeled as non-confirmable do not require sending an acknowledgment; this message type is useful for regularly sent packets, such as those containing sensor readings.
- Acknowledgment: These are responses to confirmable messages. Assuming a request is confirmable, the acknowledgment can include the response (resource) to the request; this is referred to as a "piggybacked response." If the response is not immediately available, it will be sent later in a separate message ("separate response").
- Reset: A reset message is sent in response to a request that the server cannot process due to a lack of context.

CoAP utilizes a basic binary header configuration for its communication, employing two message categories: requests and responses. The most compact CoAP message spans 4 bytes, excluding the token, options, and payload components. Subsequent to the header, the token value (ranging from 0 to 8 bytes) is followed, which could potentially be succeeded by an array of options in an optimized TLV (Type Length Value) structure. Any bytes beyond the header, token, and options (if present) are designated as the message payload. In 2.13 is depicted the structure of a CoAP message.

Octet offset		0							1							2							3														
	Bit offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31				
4	32	ver	type	token length			request/response code							message ID																							
8	64	token (0-8 bytes)																																			
12	96	options (if available)																																			
16	128	options (if available)																																			
20	160	1	1	1	1	1	1	1	1	payload (if available)																											

Figure 2.13. Structure of CoAP message (reproduced from [12])

CoAP provides support for the fundamental methods such as GET, POST, PUT, and DELETE, which can be seamlessly correlated with their counterparts in HTTP.

- GET: Retrieve the information pertaining to the representation of a specific resource, identified by the URI of the request. The GET method is secure and idempotent.
- POST: It requires a representation enclosed within a request to be processed. The operation performed by the method is determined by the server providing the resource (origin server) and depends on the "target resource." The

operation can involve creating a new resource or updating the existing target resource. The POST method is neither secure nor idempotent.

- UPDATE: It necessitates that a resource identified by a request URI be updated or created with the representation enclosed within the request. The format of the representation is specified in the "Content-Format" option. The PUT method is not secure but is idempotent.
- DELETE: It demands that a resource, identified by a request URI, be deleted. The DELETE method is not secure but is idempotent.

Chapter 3

Proposal architecture

In this chapter will be presented the proposed architecture, represented in Figure 3.1. It allows tracking of people's movements anonymously and in compliance with the current regulations regarding the processing of personal data, as regulated in Europe by the GDPR.

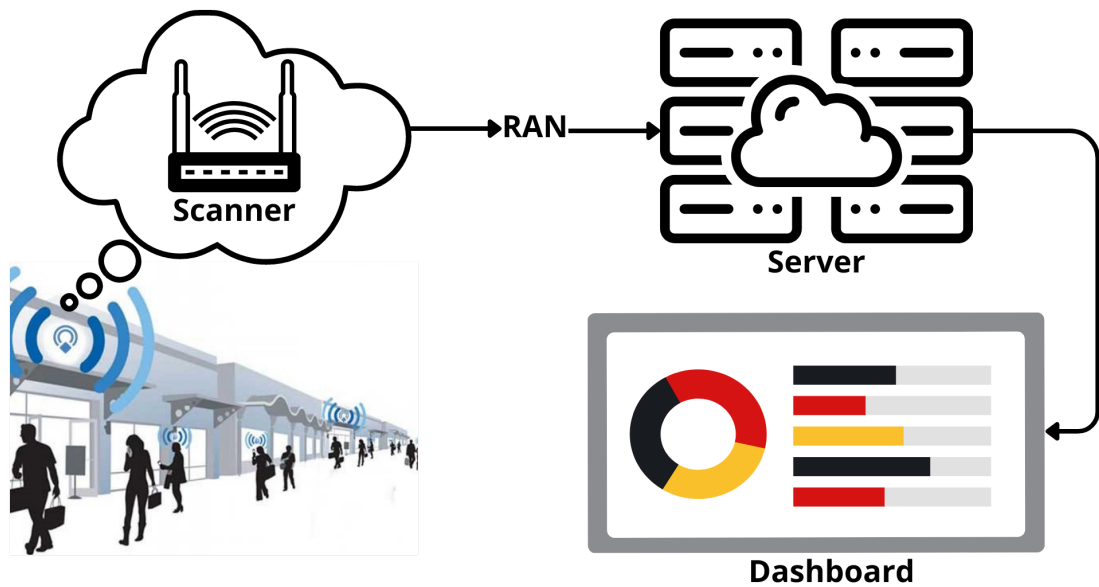


Figure 3.1. Proposal architecture

3.1 WiFi scanner

The devices used for scanning must also allow code execution. This is because data processing is carried out locally, avoiding the transmission of sensitive data over the network. Through specific analysis tools such as *tshark* or *tcpdump*, it is possible to perform scans on the 2.4GHz and 5GHz bands. Specific data useful for future steps must be extracted from the captures. Some frequently used pieces of information can include:

Devices employed for scanning

- Source Address (SA): MAC address of the message sender, which in most cases, as observed in Section 2.3.2, is a random value.
- Timestamp: The moment when the message was captured by the scanner, necessary for de-randomization operations.
- RSSI: Received Signal Power Indicator, usable for filtering operations and outlier removal.
- HT Capabilities: Set of information and parameters related to High Throughput (HT) technology for WiFi networks visible in Figure 3.2. Potentially usable to recognize messages generated by the same device but with different SAs.

```

Tag: HT Capabilities (802.11n D1.10)
  Tag Number: HT Capabilities (802.11n D1.10) (45)
  Tag length: 26
  HT Capabilities Info: 0x006f
    .... = 1 = HT LDPC coding capability: Transmitter supports receiving LDPC coded packets
    .... = 1. = HT Support channel width: Transmitter supports 20MHz and 40MHz operation
    .... = 11.. = HT SM Power Save: SM Power Save disabled (0x3)
    .... = 0 .... = HT Green Field: Transmitter is not able to receive PPDU's with Green Field (GF) preamble
    .... = 1. .... = HT Short GI for 20MHz: Supported
    .... = 1.. .... = HT Short GI for 40MHz: Supported
    .... = 0... .... = HT Tx STBC: Not supported
    .... = 00 .... = HT Rx STBC: No Rx STBC support (0x0)
    .... = 0.. .... = HT Delayed Block ACK: Transmitter does not support HT-Delayed BlockAck
    .... = 0... .... = HT Max A-MSDU length: 3839 bytes
    ..0 .... = HT DSSS/CCK mode in 40MHz: Won't/Can't use of DSSS/CCK in 40 MHz
    ..0. .... = HT PSMP Support: Won't/Can't support PSMP operation
    .0.. .... = HT Forty MHz Intolerant: Use of 40 MHz transmissions unrestricted/allowed
    0... .... = HT L-SIG TXOP Protection support: Not supported
  
```

Figure 3.2. HT Capabilities

3.2 Chain of processing

Once captured, the information needs to be processed. To achieve this, it would be advisable to create a chain of processing units, each with individual elementary tasks, in order to ensure the scalability of the system.

As mentioned earlier in Section 3.1, from the captures, it's necessary to extract useful data, and to do this, it's possible to create a compatibility layer, a specific middleware.

After the preliminary phase is completed, packets originating from the same SA are grouped, creating lists, while attempting to derive information about the network card manufacturer by examining the OUI. Simultaneously, additional data is obtained, such as the device's dwell time within the scanner's range and the variation in received signal strength. This potentially allows the determination of the means of transportation used. For each previously created list, the content is analyzed, and filtering and elimination rules are defined based, for example, on the number of occurrences or the average signal strength.

3.2.1 De-randomizer

The middle phase of the chain is the de-randomization operation, in which an attempt is made to create clusters of captured probe requests originating from the same device but with different source MAC addresses.

Considering two different MAC addresses captured by the sniffer: MAC_i and MAC_j , de-randomization algorithm starts if:

- First view of MAC_i is before last view of MAC_j .
- HT capability of MAC_i is similar to HT capability of MAC_j .

To do this, since the available information is limited, it is necessary to calculate a correlation probability between the two MAC addresses. The higher this probability is, the greater the chances that they correspond to the same device. The score represents the inverse proportion of:

- ΔT_{ij} : The time gap between the final sighting of MAC_i and the initial sighting of MAC_j serves to illustrate the uninterrupted flow of incoming received frames.
- ΔP_{ij} : The difference in sequence numbers across distinct MAC addresses within the received frames reflects the ongoing nature of frame reception. This is particularly relevant considering that the sequence number follows a cyclical incremental pattern (ranging from 0 to 4095, resetting to 0 after reaching its maximum value).

- ΔS_{ij} : The disparity in received signal power (RSSI) for each MAC address. When the gap of average received signal power between the two MAC addresses is large, it becomes more likely that these two MAC addresses are not associated with the same device.

Using the Equation 3.1 the score is computed:

$$score_{ij} = \left| \frac{1}{\Delta T_{ij}} \times \frac{1}{\Delta S_{ij}} \times \frac{1}{\Delta P_{ij}} \right| \quad (3.1)$$

where:

$$\Delta T_{ij} = t_j^f - t_i^l \quad (3.2)$$

$$\Delta P_{ij} = p_j - p_i \quad (3.3)$$

$$\Delta S_{ij} = \begin{cases} s_j^f - s_i^l & \text{if } s_j^f > s_i^l \\ 4095 - (s_i^l - s_j^f) & \text{if } s_j^f < s_i^l \end{cases} \quad (3.4)$$

defining:

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

3.2.2 Bloom filter

The penultimate phase of the chain involves inserting the output information obtained from the de-randomizer, which comprises groups of MAC addresses assimilable to the same device, into various Bloom filters. Two arrays, BF_1 and BF_2 , are indeed used, each serving a specific purpose:

- BF_1 : The elements inserted are the arithmetic averages of the MAC addresses, represented in hexadecimal format, for each defined group.
- BF_2 : The elements inserted are the individual MAC addresses.

Through the first Bloom filter, it is possible to obtain, ensuring the correctness of the de-randomization operation, the count of individual devices that have entered the scanner's range. With the second one, by intersecting multiple sets of data from different scanners, it's possible to perform the operation of tracking users' movement flows.

It should be emphasized that, to comply with the current regulations for the processing of personal data, both Bloom Filters are initially loaded with random MAC addresses. All the details of this process will be covered in Chapter 4.

3.2.3 Data Transfer

The final phase of the entire chain concerns the transmission of information from individual scanners to a server. This would then allow all flow analysis operations to be carried out anonymously by intersecting the Bloom filters, as well as displaying information on potential dashboards.

In section 2.5, we examined two different protocols, MQTT and CoAP, for data transfer. Both employ the Client/Server paradigm, but MQTT, working over TCP, supporting TLS, allowing for client state awareness, and offering three levels of QoS, is preferred for implementation.

Thanks to this protocol, the two previously created Bloom filters, along with a timestamp, will be sent to the server without the need for additional encryption mechanisms.

Chapter 4

Bloom filter for flow tracking

This chapter will delve into the ways in which Bloom filters can be effectively employed to achieve a dual purpose: safeguarding user privacy while simultaneously enabling the implementation of people counting and crowd monitoring techniques.

4.1 Tuning

To properly size a Bloom filter, it's necessary to analyze the usage context and perform appropriate assessments in order to determine the right number of bits and hash functions to use, while keeping the probability of false positives below an acceptable threshold. Failure to do so could lead to two scenarios:

- A Bloom filter that is too small would result in an increased probability of false positives, rendering it ineffective.
- A Bloom filter that is too large would lead to space wastage, which is especially critical in resource-constrained contexts.

At this juncture, with n denoting the number of elements to be inserted into the filter and m representing the bit size of the Bloom filter, what is the optimal value for k that serves to minimize the probability of false positives?

- Small k increases the fraction of 0 bits in the arrays, available for an element that is not a member of S
- Large k increases the probability of finding at least a 0 bit for an element that is not a member of S

Equation 4.1 illustrates how it is possible to determine the optimal value of "k" to be used.

$$k_{opt} = \frac{n}{m} \log(2) \quad (4.1)$$

4.2 Privacy

To assess the level of privacy ensured by a Bloom filters, work [9] has examined the extent to which an adversary can acquire information in various situations. To achieve this objective, they analyze three attack scenarios that vary in terms of the adversary's awareness. In each scenario, the adversary remains unaware of the specific elements within the stored dataset.

- **Agnostic Outsider:** In this scenario, an adversary possesses no knowledge whatsoever about the algorithms, data structures, data format, and secrets employed in the scheme. This situation could arise if, for instance, some data is inadvertently leaked by an insider, and an external adversary captures the data without any understanding of its content. In these situations, it is imperative that the data does not disclose any particulars concerning its inherent composition. However, given that data typically comes with contextual information and metadata, like data origin or file names, an adversary might attempt to gather more information about the scheme using public data or reverse engineering. Hence, the feasibility of this scenario is constrained due to practical limitations.
- **Outsider:** In this case the attacker possesses an understanding of the algorithms, data structures, and data format implemented within the scheme, but lacks awareness of any concealed secrets. This scenario commonly arises when an external adversary targets the scheme's systems and networks, potentially extracting data from an employee's laptop. In such cases, the adversary might gain access to documentation, applications, or even metadata associated with the data structure and algorithms, if accessible within the system.
- **Insider:** In this scenario, the adversary possesses knowledge of the algorithms, data structures, data format, and the shared secrets employed in the scheme, yet remains unaware of the actual entries within the data set. This situation is frequently encountered when dealing with an internal adversary, such as a curious or disgruntled employee, or when a potent external adversary targets various systems and networks.

4.2.1 Brute-Force Attack

A possible attack type is the brute-force attack, where elements are randomly generated or taken from a dictionary, and then processed to be matched against

the examined Bloom filter. For this purpose, we define the binary events A and B :

- A = Event where a query returns a positive result for a certain element x
- B = Event where the queried x exists in the original data set S

Using these events, we can define the privacy metric as follows:

$$\textit{Privacy} = P(B|A) \quad (4.2)$$

This is the probability that given a query returning a positive result for an element x , this element belongs to the set S . This probability can be rewritten using Bayes' theorem as follows:

$$P(B|A) = P(A|B)P(B)/P(A) \quad (4.3)$$

Recall that for a Bloom Filter, $P(A|B)$ is 1 as it represents the probability that a query for an element x belonging to S yields a positive result.

We can also define:

$$\textit{Privacy}_{\textit{absolute}} = P(B|A)P(A) = P(A, B) \quad (4.4)$$

This absolute metric describes the probability of correctly brute-forcing a single element by chance. This can, for example, be used to estimate how many queries will need to be executed to obtain a single correct positive match. The main difference from the first metric is that we now consider negative results as well, as $P(A)$ is the probability of a positive query result.

However, a lower $P(A)$ only affects the runtime of an attack and not directly the privacy. Nevertheless, in cases where the probability of producing a positive result $P(A)$ is infinitesimal, a brute-force attack as described above could become computationally infeasible for an adversary.

In order to assess the attack's accuracy, we need to consider the probability $P(B)$. This probability depends on both the source data set being used and the knowledge of the attacker. For this purpose, we assume that an adversary has some data set used for a brute-force attack denoted by the set S_{adv} , or some algorithm generating said data set. The size of this data set S_{adv} depends on the type of data in S , the information available to the attacker, and the assumptions made. In cases where no feasible data sets or generators of such exist, for example for schemes using names of people, an adversary can only use information that is available to them. Furthermore, in cases where a data set S_{adv} is too large for a brute-force attack, thereby generating many false positives, an adversary might choose to limit this data set by making certain assumptions. If S and S_{adv} are known, we can write the probability that a randomly queried element x exists in S as follows:

$$P(B) = P(x \in S) = |S \cap S_{adv}|/|S_{adv}| \quad (4.5)$$

If S is contained in S_{adv} , then the intersection will yield S , so knowing the size of S is sufficient to calculate the probability. However, since no adversary will have the same knowledge and skill, it's impossible to make general assumptions about the size of S_{adv} .

4.3 Privacy Protection

The performance of a Bloom filter is not solely determined by its false positive probability. In a range of applications, performance is additionally impacted by the count of elements that seem to be added to the filter due to false positives, even though they are not actually present.

As an example [7] explores the scenario of using Bloom filters as a cache memory to reduce the cost of accessing the main memory, requesting information directly to the filters. The paper illustrates that if the *a priori* probability that the element is in the cache is low enough before accessing the Bloom filter, it is always better to ignore the response from the cache and go directly to the main memory, even if the access cost is higher. We now describe how Bloom filters can be used to preserve anonymity, but before going further, we need to consider some definitions.

Definition 1

A set V is called *Hiding Set* for a Bloom filter $BF(S)$ if V contains all the elements $v_i \in U$ s.t. $v_i \notin S$ and a query for v_i returns 1. Is also possible to say that v_i is a *false positive*. An expression proposed in [4] allow to calculate the cardinality of V represented by the random variable N_v with a binomial probability distribution:

$$P\{N_v = v\} = \binom{|U| - n}{v} \psi(m, k, n)^v (1 - \psi(m, k, n))^{|U| - n - v} \quad (4.6)$$

and mean value $E[N_v] = (|U| - n)\psi(m, k, n)$. Figure 4.1 depicts an example of a *Hiding Set* where a 10-bit Bloom filter with 2 hash functions has been used to insert 3 elements (x_1, x_2, x_3). At the same time, 3 elements (v_1, v_2, v_3) show up as false positives since a query to the filter would yield a positive result.

Definition 2

An element $x \in S$ inserted in $BF(S)$ is defined *deniable* if $\forall i \in \{1..k\}$ exist at least one element $v \in V$, such that $\exists j \in \{1..k\}$ s.t. $H_i(x) = H_j(v)$. A $BF(S)$ is γ -*deniable* whenever a randomly chosen element $x \in S$ is deniable with probability γ .

Note that the definition of γ -deniability explicitly requires that the covering elements do not belong to the original set S but rather to the hiding set V . Hence,

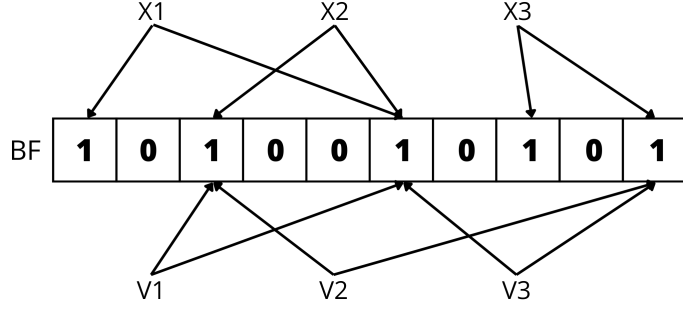


Figure 4.1. Hiding set

an element is considered deniable if it can be substituted with items that were not initially present in the Bloom filter, all while keeping the bit map unchanged.

It is also feasible to compute the size of the hiding set, as demonstrated in the study [4], using the formula:

$$\gamma(BF(S)) \simeq \left(1 - \exp\left(\frac{hk}{m(1 - e^{-kn/m})}\right)\right)^k \quad (4.7)$$

with

$$h = (|U| - n)\psi(m, k, n) = (|U| - n)(1 - e^{-kn/m})^k \quad (4.8)$$

depicting the mean number of elements in the hiding set.

Definition 3

Considering a Bloom Filter $BF(S)$ and $x \in S$ inserted in $BF(S)$, x is $K - Anonymous$ if exists at least $K - 1$ hiding set elements $\langle v_1 \dots v_{K-1} \rangle \in V$, such that $\exists \langle j_i \dots j_{K-1} \rangle \in 1..k$ s.t. $H_i(x) = H_{j_1}(v_1) = \dots = H_{j_{K-1}}(v_{K-1})$. Consequently, it is possible to say that a Bloom filter $BF(S)$ is $\gamma - K$ -anonymous if each randomly chosen element is K -anonymous with probability γ .

Using 4.9 proposed in [4] is possible to compute the $\gamma - K$ -anonymity for a specific Bloom filter configuration.

$$\gamma(K, BF(S)) \simeq \left(1 - \exp\left(-\frac{hk}{m(1 - e^{-kn/m})}\right) \sum_{i=0}^{K-2} \frac{(hk/[m(1 - e^{-kn/m})])^i}{i!}\right)^k \quad (4.9)$$

4.3.1 Anonymization Noise

For the reasons outlined in section 4.2, and specifically in section 4.2.1, it is necessary to safeguard the privacy of the information inserted in a Bloom filter through appropriate methodologies. In the context at hand, the filter will be used to store

MAC addresses, which are considered sensitive data under the GDPR. Therefore, we propose the utilization of the γ -deniability property, as defined in section 4.3, to ensure the ability to deny the membership of all inserted elements. However, as specified in 4.3, γ represents the probability of being able to deny the presence of a randomly chosen element, using another element that is not actually part of the set S . To guarantee this possibility for all elements present in the Bloom filter, it's necessary for the value of γ to reach its maximum, which is 1. This way, any inserted element can be denied.

Initially, however, this probability is 0. In fact, when the Bloom filter is completely empty and the configuration parameters m and k , which are constants, are ignored, the value of n (equal to 0) will nullify the equation 4.8, which will in turn set the value of 4.7 to 0. To increase the value of γ , it is necessary to insert elements into the Bloom filter and after a certain number of insertions, γ will reach 1.

However, to protect privacy, it's not possible to use MAC addresses captured before γ reaches its maximum value. For this reason, we introduce the concept of *anonymization noise*. This noise consists of n_{min} randomly generated elements that are not subject to protection constraints. They should be inserted as soon as the Bloom filter is created to bring the value of γ to 1.

4.3.2 Multiple Anonymity

In section 4.3.1, a method for ensuring the privacy of insertions into a Bloom filter was analyzed using the definition 4.3. However, it's important to consider that this approach covers an actually inserted element with only one element not actually present.

To further enhance the provided protection, it's possible to leverage the definition 4.3. By choosing a value of K and maximizing the expression 4.9 (thus bringing the value to 1), a coverage of at least $K - 1$ elements not actually inserted can be achieved, rather than just a single element.

In this situation as well, similar to the one described earlier, it is essential to employ "anonymization noise" to ensure that from the very first insertion of genuine MAC addresses, each of them is protected by at least $K - 1$ elements.

Practical Examples

Let's now examine two examples where we can observe the difference between a Bloom filter with γ deniability of 1 and one with γ deniability of 0.66. In both figures (Figure 4.2 and Figure 4.3), the elements x_1, x_2, x_3 have been inserted into the filter, while the components of the hiding set are v_1, v_2, v_3 .

In Figure 4.2, any element randomly selected from S can certainly be denied. For instance, element x_1 can be denied using a combination of v_1 and v_2 , while element x_3 can be denied using v_1, v_2 , and v_3 .

Conversely, in Figure 4.3, there is a 0.66 probability of randomly selecting a deniable element. This is because the element x_3 cannot be denied by any other element belonging to the hiding set.

	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]	B[8]	B[9]
X1	1	0	1	0	0	0	0	1	0
X2	0	0	1	1	0	0	0	1	0
X3	0	0	0	1	0	1	0	0	1
BF	1	0	1	1	0	1	0	1	1
V1	1	0	1	1	0	0	0	0	0
V2	0	0	1	0	0	1	0	0	1
V3	1	0	0	0	0	1	0	1	0

Figure 4.2. Bloom filter with $\gamma = 1$ composed of elements (x_1, x_2, x_3)

	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]	B[8]	B[9]
X1	1	0	1	0	0	0	0	1	0
X2	0	0	1	1	0	0	0	1	0
X3	0	0	0	1	0	1	0	0	1
BF	1	0	1	1	0	1	0	1	1
V1	1	0	1	1	0	0	0	0	0
V2	0	0	1	0	0	1	0	1	0
V3	1	0	0	0	0	1	0	1	0

Figure 4.3. Bloom filter with $\gamma = 0.66$ composed of elements (x_1, x_2, x_3)

In the final example shown in Figure 4.4, we can observe the case of a Bloom filter with $\gamma = 1$ and $K = 3$. This means that any randomly chosen element will have each of its bits covered by at least $K - 1$, so 2 elements from the hiding set. For instance, the element x_1 will have bit $B[1]$ covered by v_1 and v_3 , bit $B[3]$ covered by v_1, v_2 , and v_4 , and bit $B[8]$ covered by v_2, v_3 , and v_4 .

	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]	B[8]	B[9]
X1	1	0	1	0	0	0	0	1	0
X2	0	0	1	1	0	0	0	1	0
X3	0	0	0	1	0	1	0	0	1
BF	1	0	1	1	0	1	0	1	1
V1	1	0	1	1	0	0	0	0	0
V2	0	0	1	0	0	1	0	1	1
V3	1	0	0	0	0	1	0	1	0
V4	0	0	1	1	0	0	0	1	1

Figure 4.4. Bloom filter with $\gamma = 1$ and $K = 3$ composed of elements $(x1, x2, x3)$

4.4 Counting

Now let's consider the scenario in which multiple WiFi sensors for device counting are deployed across a large area. Each sensor will store the collected MAC addresses within a local Bloom filter, enabling two main operations:

- Counting the number of devices that have passed through the coverage area of that sensor.
- Anonymously identifying patterns of people's movement.

The simple task of counting elements within a Bloom filter can be accomplished using the formula shown in equation 4.10.

$$c_1 = -\frac{m}{k} \ln \left(1 - \frac{t}{m} \right) \quad (4.10)$$

By knowing the common MAC addresses between different Bloom filters, it is possible to identify the people's movement patterns performing intersections.

4.4.1 Intersection

In addition to individual element operations, Bloom filters can be used to perform set unions and intersections. Let $S1$ and $S2$ be subsets of U , represented by the Bloom filters $BF(S1)$ and $BF(S2)$, which use the same configuration parameters. To calculate the intersection, it is necessary to perform a bitwise logical AND operation between the two vectors. Once the Bloom filter of the intersection is

obtained, it will be possible to calculate the number of elements contained within it using two different equations. The first formula is shown in (4.10) (which should be replaced with t_3 instead of t), and the second formula is proposed in the work [6] and shown in (4.11).

$$c_2 = \frac{\ln\left(m - \frac{t_3 \times m - t_1 \times t_2}{m - t_1 - t_2 + t_3}\right) - \ln(m)}{k \times \ln\left(1 - \frac{1}{m}\right)} \quad (4.11)$$

where t_1 represents the number of bits equal to 1 in BF_1 , t_2 represents the number of bits equal to 1 in BF_2 and t_3 represents the number of bits equal to 1 in BF_3 .

Chapter 5

Implemented Solution

This chapter will present the implemented solution designed to enable the safe management of people in crowded areas. In this context, the *TrialsNet* project (TRials Supported By Smart Networks Beyond 5G), among its various objectives, aims to collect and process data during outdoor public events to enable participant counting, flow analysis, and density assessment, ultimately enhancing collective safety. All implementation choices made will be explained, and finally, we will focus on the data processing workflow.

5.1 Hardware

To implement our proposal on a large scale, the primary factors to consider are the cost and the performance; thus, we strive to find an optimal balance between these two variables. Hardware that is too budget-friendly and offers low performance would prove ineffective and result in significantly prolonged execution times, whereas extremely costly and highly performant hardware might entail a resource waste.

In our scenario, the most suitable choice is leveraging a Single Board Computer (SBC), which optimizes the trade-off between costs, power consumption, and space utilization. As for the software aspect, it is preferable to opt for a Linux-based operating system, given that Linux supports a wide array of tools and is easily configurable through the command line. Concerning hardware components, it would be ideal to have a multicore processor capable of managing multiple tasks simultaneously, coupled with a sufficiently ample amount of RAM to ensure the system's proper operation. Furthermore, it would be necessary to include an LTE dongle for internet connectivity, thereby enabling data transmission without the need for a dedicated network, and a USB WiFi modem for capturing probe requests, as it's not possible to use the integrated WiFi interface in monitor mode. Regarding power supply, we should not encounter substantial limitations, as the board

will be powered through the standard electrical grid. However, for temporary and short-term deployments, it's also possible to use a power bank for power supply if needed.

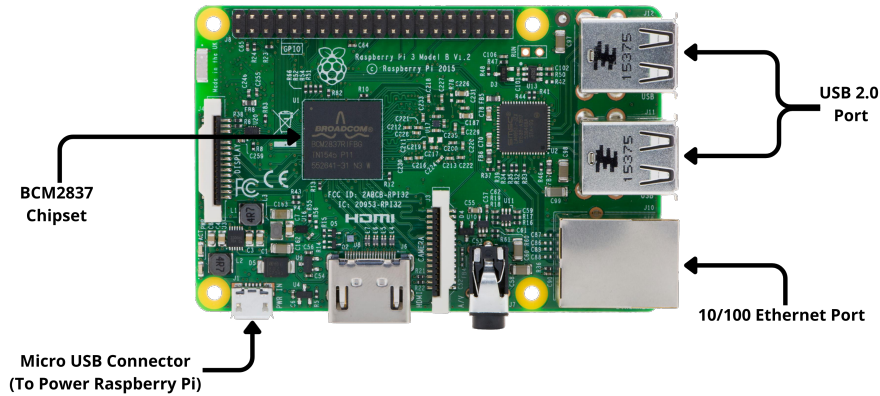


Figure 5.1. Raspberry Pi 3 Model B

After conducting appropriate market evaluations and considering its widespread use, which ensures a large supportive community in case of issues, the choice fell on the Raspberry Pi 3 Model B, as shown in Figure 5.1. It provides a Broadcom BCM2837 1.2GHz Quad-Core ARM Cortex-A53 processor, 1GB of LPDDR2 RAM, 4 USB 2.0 ports, and a Micro USB connector with a 5V/2.5A voltage/current output.

Due to the fact that the integrated WiFi interface of the Raspberry Pi does not have the capability to operate in monitor mode, we needed to obtain a USB WiFi dongle. During the capture phase, the USB dongle will be active, while the built-in interface will be disabled to prevent potential interference. According to the 802.11 standard, there are two typical frequency bands for WiFi: 2.4 GHz and 5 GHz. However, since sniffing can only be performed on one band at a time, we chose to use the most common 2.4 GHz band. This decision was made to ensure greater compatibility with all possible devices that might pass through the covered area.

Furthermore, the presence of a reliable server is also essential for the transmission of the gathered and processed data. With the assistance provided by the Politecnico di Torino, we gained the opportunity to employ a system equipped with a publicly accessible IP address, which allowed us to establish an MQTT broker along with a linked database.

Lastly, as mentioned earlier, it was essential to integrate an LTE dongle in order to enable remote control of the system and facilitate data transmission. The acquired USB device provides the option to use a data SIM card compatible with 4G technology.

A depiction of the system architecture can be observed in Figure 5.2

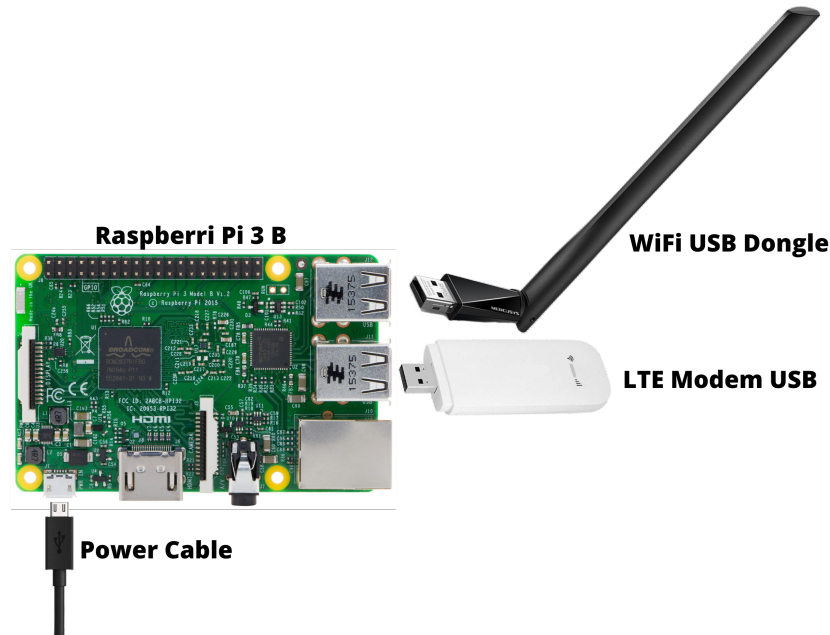


Figure 5.2. Raspberry Pi 3 Model B Configuration

5.2 System Description

In the operational context, the Raspberry Pi is powered by the electrical grid with the support of IREN, and the WiFi and LTE dongles are connected and operational. The integrated WiFi card will remain consistently deactivated during the data capture phases. As soon as the brief configuration phase is completed, the two main processes are initiated: data capture and data processing. Both are part of an infinite cycle, as the sensor must constantly capture and process information.

5.2.1 Remote Accessing to Raspberry

To enable remote access to the Raspberry Pi, two different solutions have been employed based on specific requirements. It's noteworthy to consider that, since the device is connected to the internet via the LTE dongle, it cannot have a direct static IP address due to NAT limitations, and furthermore, the firewalls of various Internet Service Providers do not allow direct SSH connections. The two adopted solutions are RealVNC and ZeroTier.

RealVNC [10] is a remote access application that enables viewing and interact with a computer’s desktop from another device, such as a laptop, smartphone, or tablet. RealVNC is compatible with various platforms and operating systems, including Windows, macOS, Linux, and mobile devices.

ZeroTier [10] One is a free and open-source software application that incorporates cutting-edge advancements in Software-Defined Networking (SDN) to enable users to establish secure and easily controllable networks, simulating a scenario where interconnected devices are located in the same physical vicinity. ZeroTier offers a user-friendly web console for efficient network administration and deploys endpoint software for clients. It employs encrypted Peer-to-Peer technology, which distinguishes it from conventional VPN solutions. Unlike traditional setups, where communications are relayed through a central server or router, ZeroTier enables direct peer-to-peer messaging between hosts.

5.2.2 Capturing WiFi Probe Request

The sniffer is responsible for detecting passersby’s probe requests and temporarily saving them to a local file using the tshark tool. By using this tool, it’s possible to selectively filter all captured packets and save only the probe requests, identified by the subtype 0x04. In addition to this, the tshark command allows associating a timestamp and the received signal strength with each packet, which will be used later during the processing phase. The capture activity is performed cyclically and operates with time windows ranging from 1 to 3 minutes, depending on the selected configuration parameters.

5.2.3 Bloom Filter Dimension

As we will see later in section 5.3, two Bloom filters are employed. To determine their sizes, we apply the formula described in 4.1. In this case, since the filter will be changed approximately every 60 – 180 seconds, we consider a maximum flow of about 1000 MAC addresses, setting $m = 1000$. On the other hand, to obtain the optimal value of k to use, we still need to define the value of n , which is set to 10000 for experimental reasons. With the established values of m and n , we can calculate k as follows:

$$k_{opt} = \frac{10000}{1000} \log(2) \simeq 7 \tag{5.1}$$

5.3 Python Chain Of Processing

The information processing chain is comprised of a series of cyclically executed Python scripts as visible in Figure 5.3. Their coordination and information exchanged are managed through UNIX pipes. When a file completes its execution, it

writes to STDOUT (standard output), and the subsequent file reads the information from STDIN (standard input). In our case, we use multiple pipes to connect the scripts in sequence, and each operates independently from the others. The chain is composed as follows:

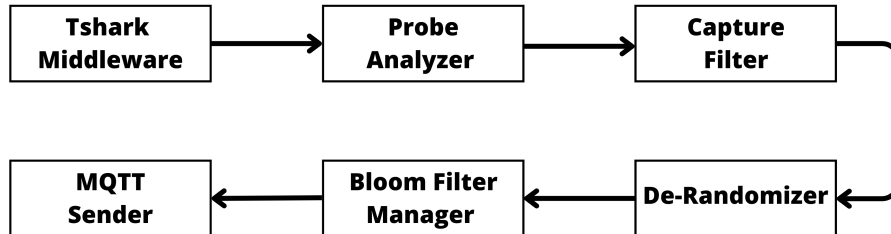


Figure 5.3. System Python Chain of Processing

- **Middleware:** From the specific folder containing the **.pcap** files to be analyzed, it reads their contents and extracts the necessary information. To perform this operation, the scapy library [8] is used, allowing access, reading, and extraction of relevant data from **.pcap** files. For each probe request present in the file, it creates a line where fields are separated by commas. During this process, integrity checks are carried out on sections such as HT Capabilities, A-MPDU Parameters, and HTEX Capabilities to verify their presence, and if necessary, replace the value with a null value. Once the entire file has been analyzed, the file itself is discarded, and the output, consisting of a list of lines called List1, is written to STDOUT.
- **Probe Analyzer:** It reads the List1 from STDIN and analyzes it to create a new list, List2. Inside List2, for each found MAC Address, the following details are included:
 1. Number of occurrences
 2. First and last sight of the timestamp
 3. First and last sequence number
 4. Maximum and minimum received signal strength, along with a calculated average
 5. Identification of the OUI associated with the MAC address through a list stored in the Raspberry's memory. If the lookup operation doesn't yield results, "Unknown" is inserted.

At this point, List2 is written to STDOUT.

- **Capture Filter:** It reads List2 from STDIN and remove packets that have a MAC occurrence count less than 2. This value has been determined based on the analyses presented in Chapter 6. Subsequently, store the refined list in List3 and print its contents on STDOUT.
- **De-Randomizer:** From List3 acquired from STDIN, a recursive de-randomization algorithm is applied to create groups of MAC addresses potentially belonging to the same device. These groups consist of pairs of MAC addresses, and for each pair, the conditions described in section 3.2.1 are checked to assign a score. A higher score indicates a greater likelihood that the two MAC addresses are associated with the same device. Consequently, the output consists of a new list, List4, composed of groups of correlated MAC addresses.
- **Bloom Filter:** From the List4 acquired from STDIN, the groups of MAC addresses associated with the same device are individually extracted and simultaneously, 2 Bloom filters are initialized, namely *BF1* and *BF2*, populated with 30 random MAC addresses to ensure privacy, as explained in section 4.3.1. For each group, the arithmetic mean is calculated and subsequently inserted into *BF1*. At the same time, individual MAC addresses are inserted into *BF2*. The reason for this is that *BF1* will be used for counting purposes, while *BF2* will be employed for an anonymous study of user flows. Each pair of Bloom filters is associated with a timestamp, and the results are printed to STDOUT.
- **Sender:** In the final phase, the pair of Bloom filters associated with the timestamp at the time of capture is sent via MQTTS (MQTT over SSL to enhance security) to the server. Each scanner will have an identifier, and the message will be sent to the "bloomfilter/*scanner_id*" topic.

Furthermore, a flowchart depicting the software-side operation of the system is displayed in Figure 5.4

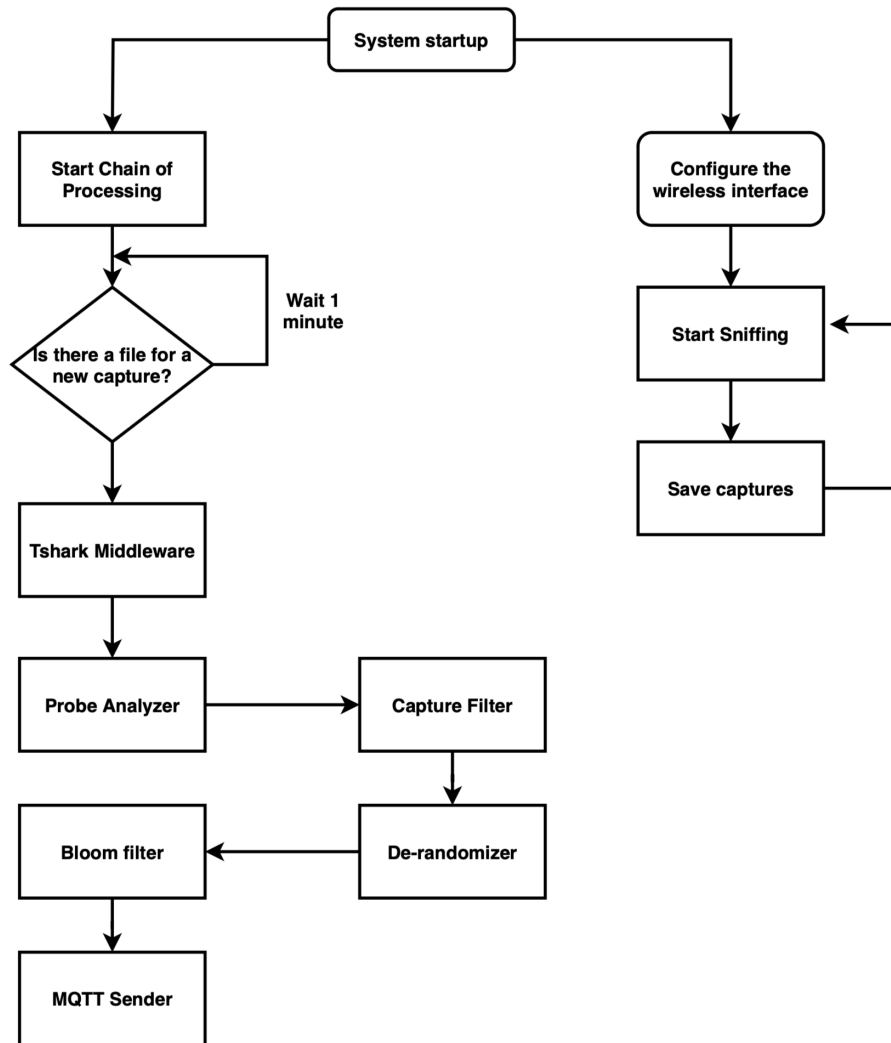


Figure 5.4. System Flow Chart

Chapter 6

Experimental evaluation

6.1 Profiling

The profiling activity is essential to evaluate the operational effectiveness of the system. As previously mentioned, the system consists of two main scripts. The first one performs simple captures using the "tshark" command, which occupies 0.1% of the CPU and 0.3% of the RAM. The second script, on the other hand, orchestrates the sequence of processes for information processing and involves the execution of four Python scripts. Since a high-level programming language is being used, it is crucial to monitor both the CPU and RAM load in order to determine if the system is capable of simultaneously executing all the scripts present.

In this context, it is relevant to note that the utilized processor is quad-core, and to carry out proper measurements, we make use of the "htop" command, available in the LINUX environment. This command provides a dynamic and real-time representation of the system's operational state, offering both a summary of system information and a list of processes currently managed by the Linux kernel.

To conduct the measurements, four acquisition tests of 60 seconds each were executed, followed by subsequent processing. Our system demonstrates an appropriate load, with peaks of 100% on the CPU and 20% on the RAM.

The system experiences spikes in CPU load as it strives to execute the scripts as quickly as possible. Each Python script in the sequence exhibits a different CPU and RAM load.

Figures 6.9 and 6.10 show graphs representing the load on the CPU and RAM over time during the execution of the most demanding script, which manages the insertion of information into Bloom filters. As can be seen, the values for RAM usage are the highest in the entire pipeline, while those for CPU are within the norm.

Figures 6.1 and 6.2, on the other hand, show the usage of CPU and RAM during the initial phase of the pipeline when the *.pcap* files are read for processing. In this case, CPU usage is very high, while RAM usage is below average.

Finally, Figures 6.3,6.4,6.5,6.6,6.7, 6.8 depict graphs of CPU and RAM usage by the analysis scripts (Probe Analyzer), filtering (Filter), and derandomization. As you can see, the filtering operation requires the least resources in terms of both CPU and RAM, while the other two operations remain within the average range.

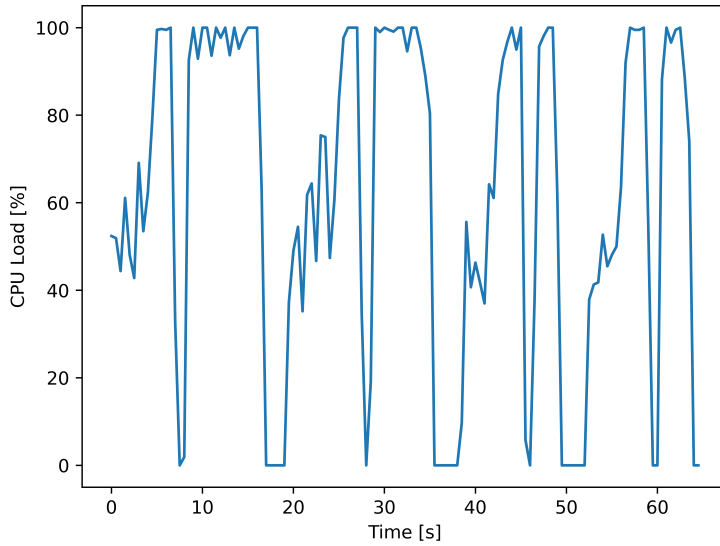


Figure 6.1. Middleware Script CPU Load Evolution Over Time

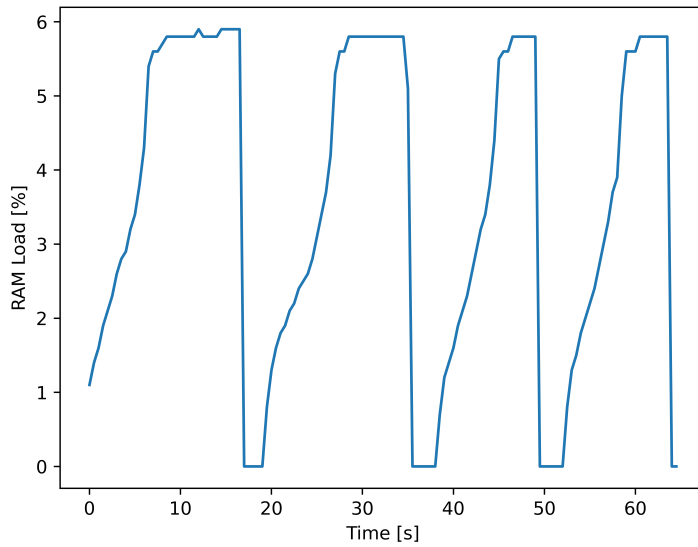


Figure 6.2. Middleware Script RAM Load Evolution Over Time

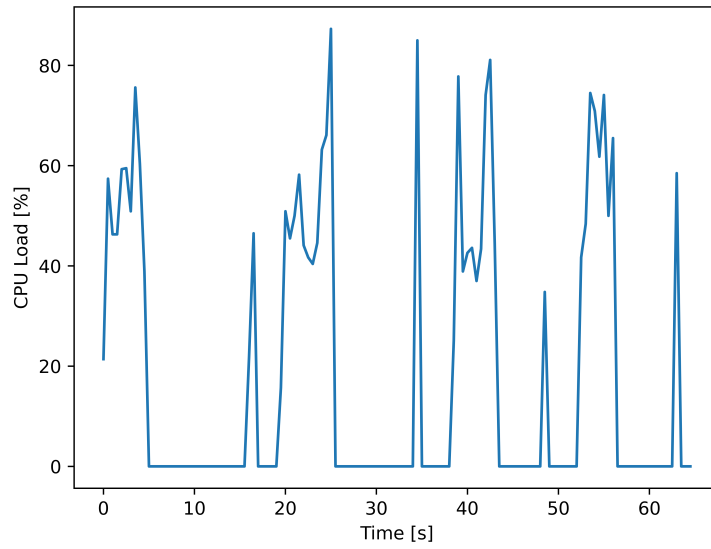


Figure 6.3. Analyzer Script CPU Load Evolution Over Time

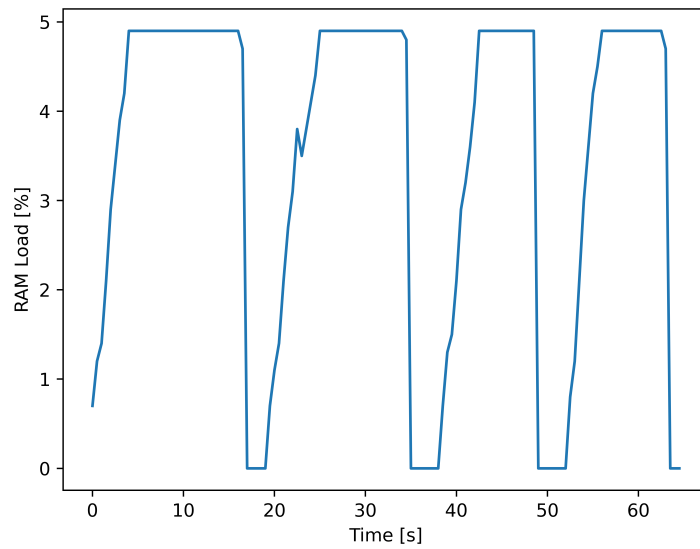


Figure 6.4. Analyzer Script RAM Load Evolution Over Time

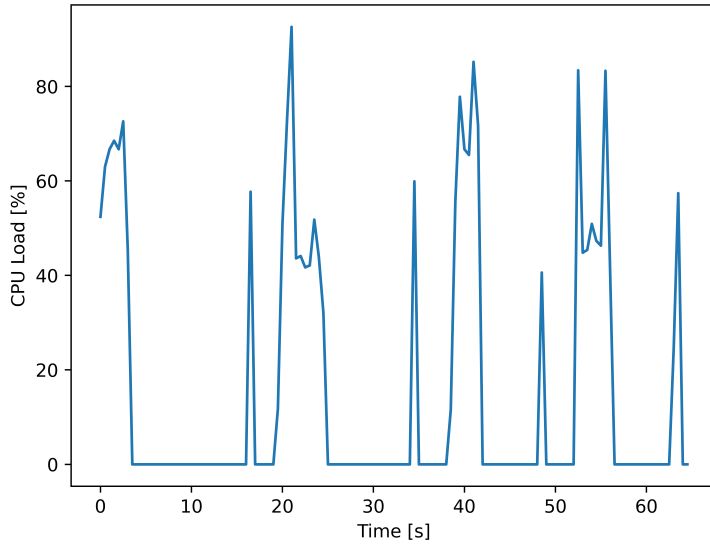


Figure 6.5. Filter Script CPU Load Evolution Over Time

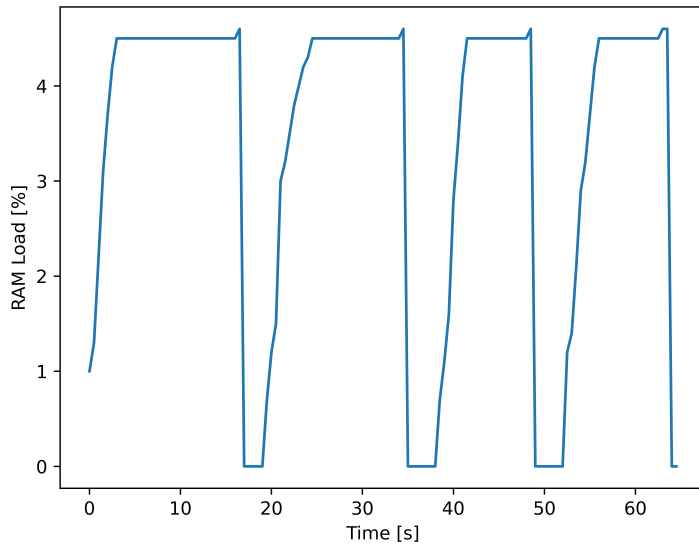


Figure 6.6. Filter Script RAM Load Evolution Over Time

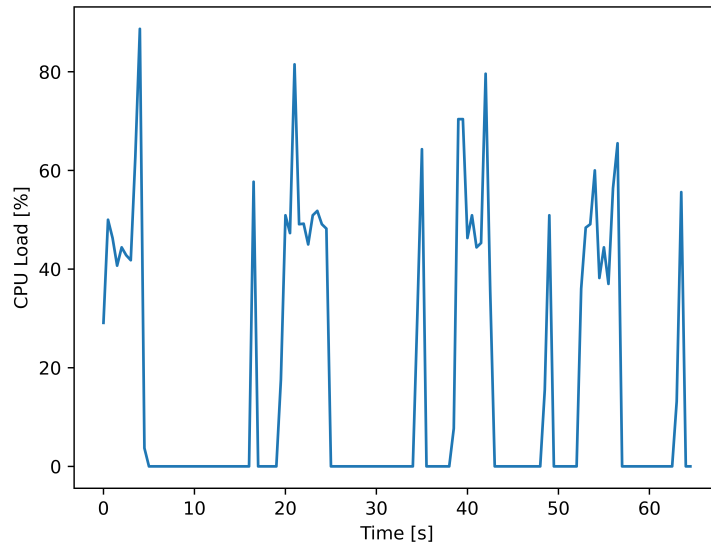


Figure 6.7. Derandomizer Script CPU Load Evolution Over Time

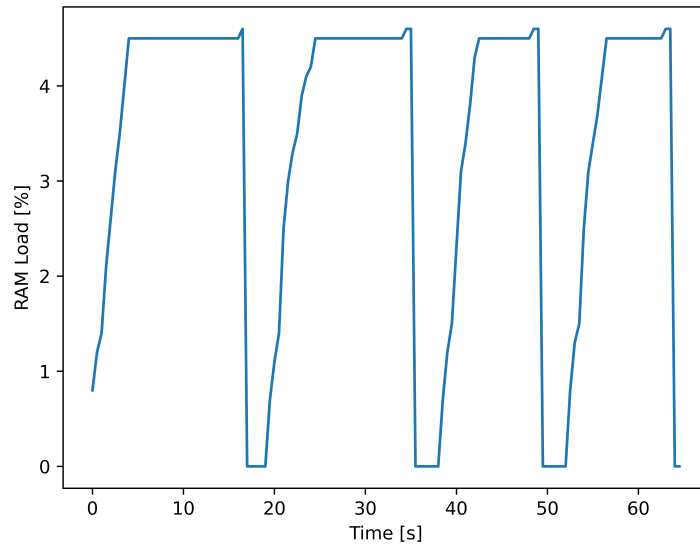


Figure 6.8. Derandomizer Script RAM Load Evolution Over Time

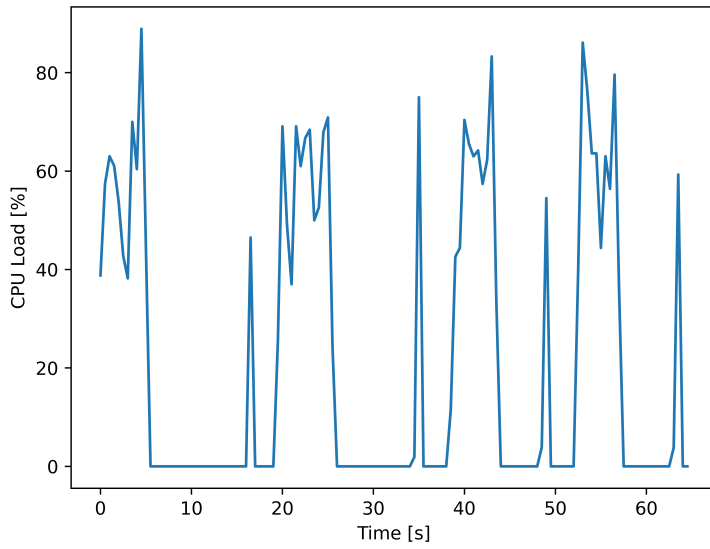


Figure 6.9. Bloomfilter Script CPU Load Evolution Over Time

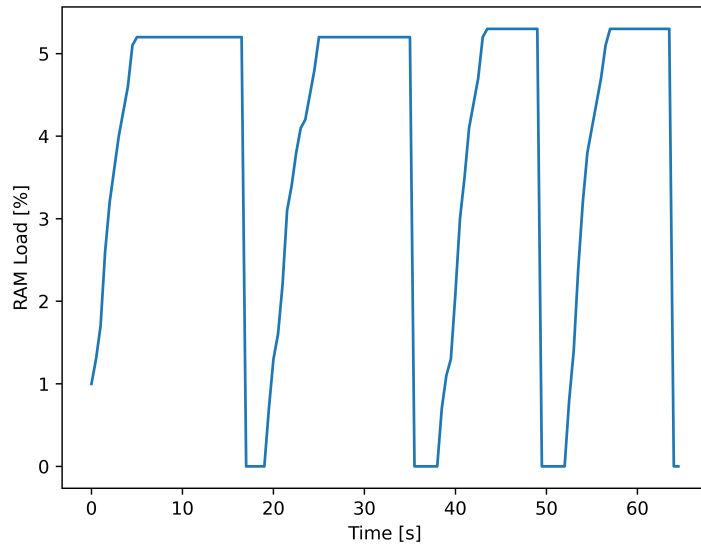


Figure 6.10. Bloomfilter Script RAM Load Evolution Over Time

6.2 Bloom filter dimension

The first task to carry out is appropriately sizing the Bloom filters for usage. As indicated in Section 4.1, it's essential to select the right size to avoid waste on one side and an increase in the false positive probability on the other. In our case, establishing two fundamental parameters was crucial: the size of the filter and the quantity of information to be inserted.

Since the Bloom filter would be changed approximately every 60 seconds, ensuring the ability to perform around 1000 insertions was of vital importance. For the size, it was agreed that an appropriate size in terms of space would be 10000 bits, or 1.25 kilobytes.

After defining the parameters m and n , it then became possible to determine the value of k through the equation in 4.1. In our case, the optimal value is $k = 7$.

In Figure 6.11, it is possible to observe the variation of the false positive probability of a Bloom filter with a configuration of $k = 7$ and $m = 10000$ during the insertion of elements. It is evident that the false positive probability remains stable until 1000 insertions, after which a phase of increasing monotony begins.

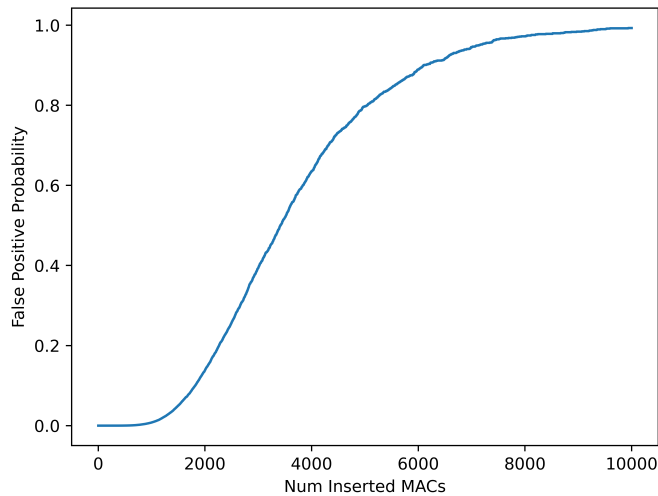


Figure 6.11. Evolution of false positive probability during the insertion of elements into a Bloom filter

In the Figure 6.12, instead, the result of an analysis on how the factor k affects the probability of false positives is shown. Taking a Bloom filter with m equal to 10000, the trend of the false positive probability has been plotted in relation to the number of addressed MACs, varying k . Four different values of k have been used. It is noticeable that higher values of k require fewer insertions to achieve the same

probability of false positives.

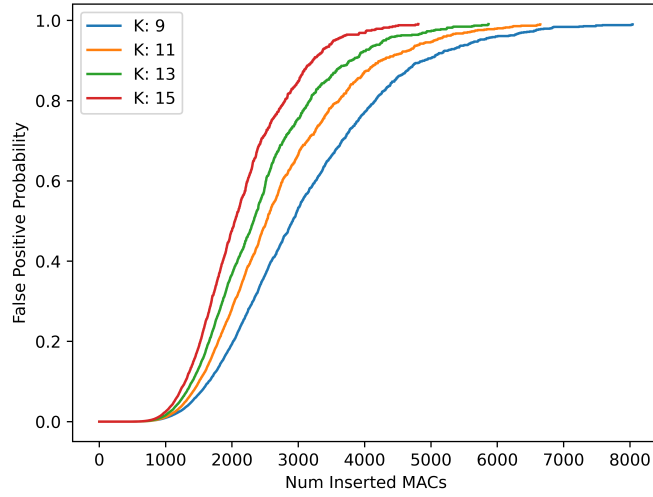


Figure 6.12. Evolution of false positive probability during the insertion of elements into different Bloom filters. Each filters is configured with $m = 10000$.

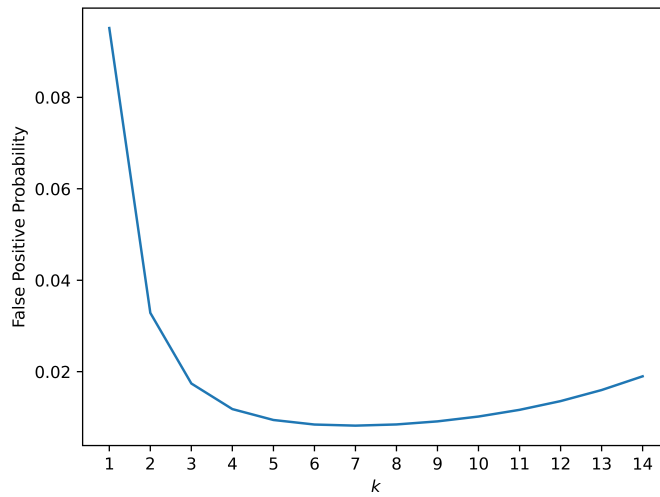


Figure 6.13. Evolution of false positive probability with different values of k .

To confirm what was previously stated regarding the value of k in our situation,

the image 6.13 shows the variation of the false positive probability in relation to the change of parameter k . For the creation of this graph, the equation 2.1 was utilized, in which the parameters m remained fixed at 10000, and n was maintained at 1000. As it can be observed, the point of minimum is obtained at $k = 7$.

6.3 Counting

As previously mentioned, our system is capable of performing counts on both individual Bloom filters and following intersections for counting common elements.

The initial measurements were conducted to assess the accuracy of counts generated by the formula indicated in 4.10. More in detail the configuration of the Bloom filter consist in $k = 7$ and $m = 10000$, initially empty. A total of 10000 elements were inserted into this filter, well beyond the value of n for which it was designed. After each insertion, relation 4.10 was used to produce a count of the quantity of contained elements, and the result was subsequently compared to the actual number of elements present. In the Figure 6.14, the result of the conducted analysis is presented. As it can be observed, the count is not significantly overestimated or underestimated, even though the number of insertions exceeds the value for which the filter was originally designed.

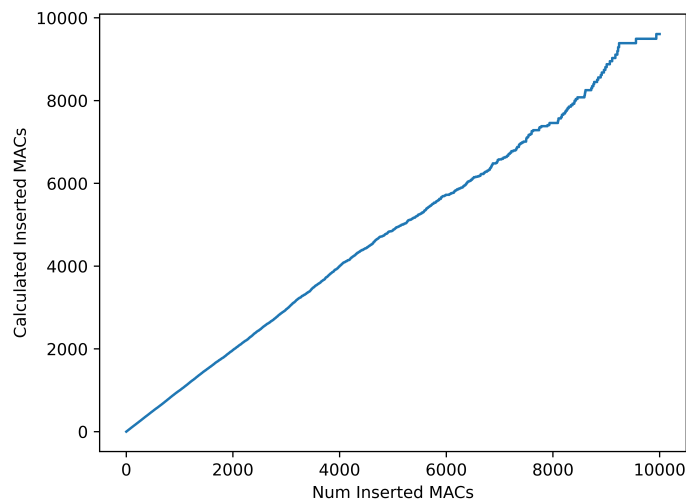


Figure 6.14. Comparison between the number of elements actually inserted and the counted ones.

This observation is further confirmed by figure 6.15, in which the generated relative error is displayed. This error remains stable until 5000 insertions, after

which it experiences a peak around the 8000 inserted elements. Nevertheless, it's important to note that the maximum value reached by the relative error is 0.08.

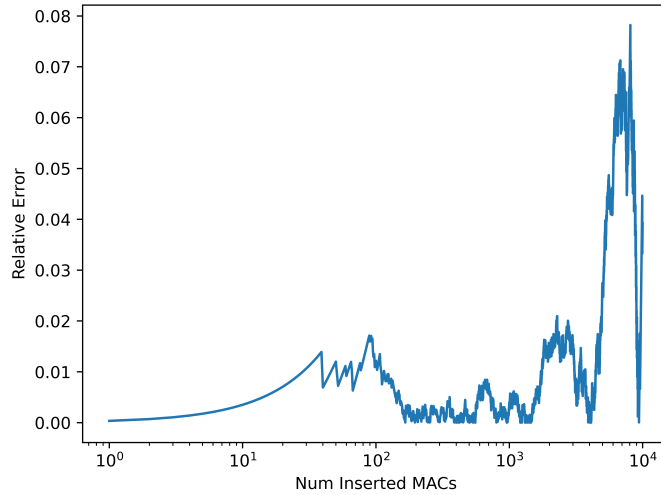


Figure 6.15. Relative error comparison between the number of elements actually inserted and the counted ones.

6.3.1 Counting intersection

As our goal is also to enable tracking and counting, anonymously, of devices passing through and detected by various scanners, we conducted comparisons to evaluate which of the proposed formulas, namely 4.10 and 4.11, is the most suitable. The measurement was conducted using two Bloom filters, namely BF_1 and BF_2 , with the same configuration, i.e. $k = 7$ and $m = 10000$. Starting with two completely empty filters, the modulus operandi was as follows:

1. Insert 500 random MAC addresses into BF_1 .
2. Insert 500 random MAC addresses into in BF_2 .
3. Insert 1 identical MAC address into both BF_1 and BF_2 .
4. Calculate the Bloom filter BF_3 from the intersection of BF_1 and BF_2 , as described in 4.4.1.
5. Count the elements that are part of the intersection using the two previously indicated formulas.

6. Return to step 3 until 500 MAC addresses are inserted.

In Figure 6.16, it is possible to observe the obtained result. Indeed, it can be noted that the two curves are nearly parallel, even though the formula 4.10 consistently tends to overestimate the number of elements present in the intersection.

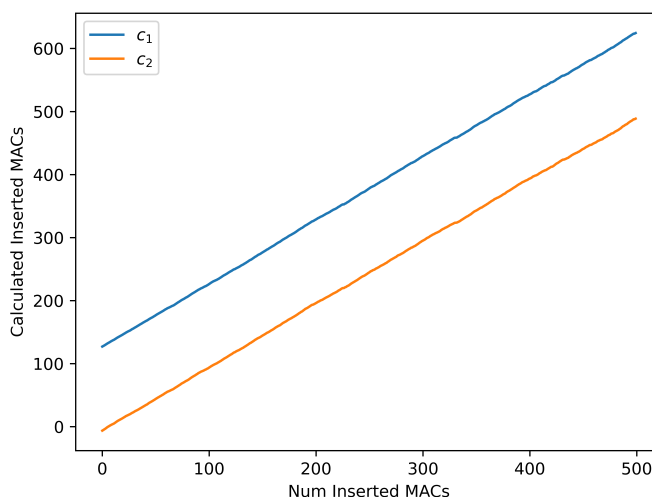


Figure 6.16. Comparison between the estimators c_1 and c_2 to evaluate the number of people moving from one scanner to the other

To confirm what was previously stated, Figure 6.17 shows the comparison between the relative errors of the two formulas. The relative error for c_1 is noticeably high when dealing with a small number of stored MAC addresses, becoming acceptable (less than 100%) only when more than 100 MAC addresses are stored. Conversely, the relative error of c_2 remains quite low even with a small number of stored MAC addresses. This outcome aligns with findings in [6], indicating a consistent pattern. However, it underscores a different balance among accuracy, complexity, and privacy in determining the count of MAC addresses within the intersection of BF_1 and BF_2 .

Specifically, (4.10) only necessitates knowledge of the number of ones in BF_3 , while (4.11) (although highly accurate) demands knowledge of the number of ones in BF_1 , BF_2 , and BF_3 on the same server. Consequently, these two solutions exhibit varying levels of adaptability to a given architecture, whether it is centralized or distributed. We leave the exploration of the optimal balance among accuracy, implementation complexity, and privacy-preserving architectures for future research.

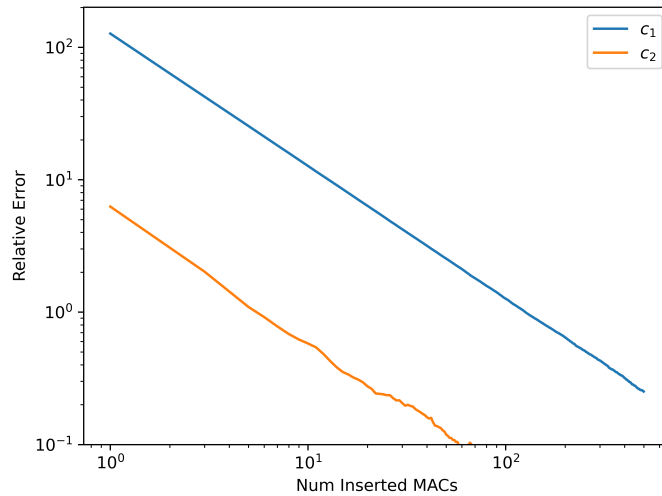


Figure 6.17. Relative error of the estimators c_1 and c_2 to evaluate the number of people moving from one scanner to the other

6.4 Anonymity

In Section 4.3.1, we introduced a method to ensure the anonymity of elements inserted into a Bloom filter by leveraging the coverage of other elements present in the *Hiding Set*. However, to protect the anonymity of the first inserted elements, we introduced the concept of "anonymization noise," represented by a set of n_{min} randomly generated MAC addresses, to ensure that γ is already equal to 1 from the very first true insertion. To determine the value of n_{min} , we used the relation 4.7 applied to a Bloom filter with configuration $k = 7$ and $m = 1000$. Starting from an empty filter, we began inserting one random MAC address at a time and subsequently calculated the value of γ . The result, visible in Figure 6.18, indicates that for the configuration we adopted, approximately 30 insertions are sufficient to achieve a γ value of 1.

For this reason, as soon as a new Bloom filter is instantiated, it will be immediately populated with at least 30 random MAC addresses in order to ensure the privacy of insertions, without the need to resort to the use of cryptographic functions that would burden the system, as well as introduce additional constraints related to the protection and management of encryption keys.

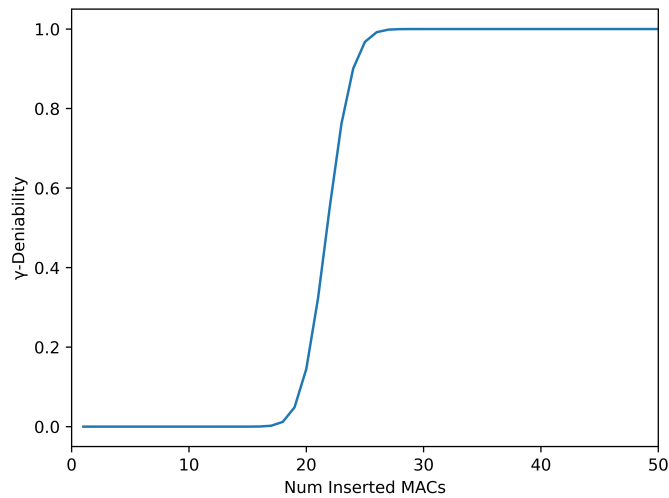


Figure 6.18. γ -deniability value in relation to the number of inserted MAC addresses

6.4.1 Multiple Anonymity

In Section 4.3.2, we discussed the possibility of ensuring coverage for each element inserted into a Bloom filter by a variable number, denoted as K , of other elements belonging to the Hiding Set. Similar to what was done in Section 6.4, once a value for K is fixed, it becomes necessary to establish the value of n_{min}^K , so that for each element randomly extracted from the filter, there are $K - 1$ non-inserted elements capable of covering it. To achieve this, we took a Bloom filter with $m = 10000$ and $k = 7$, applying the relation 4.9 for various values of K . The result is visible in Figure 6.19.

Please note that by increasing the number of initially loaded random MAC addresses in the Bloom filter by just 10, it's possible to ensure anonymity of order $K = 20$. This means that, for any given element, we would always be able to cover it with 19 elements belonging to the Hiding Set.

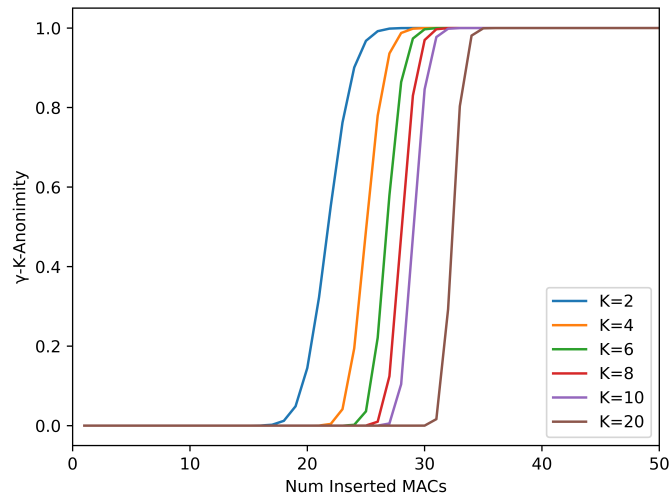


Figure 6.19. γ -deniability value, for different K, in relation to the number of inserted MAC addresses

Chapter 7

Conclusion and Future Work

This thesis proposes an automatic and anonymous system for people counting leveraging WiFi probe requests received from mobile devices and the use of Bloom filters for storage. Nowadays, developments in safeguarding individuals' privacy are continuously evolving, from the basic enabled random MACs on most devices to the GDPR regulations published in 2016 and enacted in 2018. For this reason, we have developed a method that leverages the deniability and anonymity properties of Bloom filters to preserve user privacy.

The results are very positive, especially in terms of coverage of the inserted elements, as it is possible to arbitrarily choose a value of K to define the minimum number of elements that will ensure coverage for each element in the filter. Utilizing this methodology allows us to store MAC addresses within Bloom filters without the need for additional cryptographic functions for protective purposes, significantly enhancing ease of deployment and usage. As mentioned earlier, this completely frees us from the obligation to manage encryption keys and all related requirements.

Undoubtedly, this thesis work could serve as a starting point for more extensive projects. For example, in the context of commerce, it would be useful to analyze how long a user visits specific stores to provide more relevant information about their preferences or interests. Similarly, in an airport or railway context, studying the busiest areas could enhance safety in the area. Finally, in an urban planning context, it would be valuable for authorities to understand how people move within a city to design roads, public transportation, parking, and infrastructure more efficiently.

Bibliography

- [1] Probe request frame. [Online]. URL: <https://www.oreilly.com/library/view/80211-wireless-networks/0596100523/ch04.html>.
- [2] Android. Mac randomization. [Online]. URL: <https://source.android.com/docs/core/connect/wifi-mac-randomization?hl=en>.
- [3] Apple. Mac randomization. [Online]. URL: <https://support.apple.com/it-it/guide/security/secb9cb3140c/web#>.
- [4] Giuseppe Bianchi, Lorenzo Bracciale, and Pierpaolo Loretì. "better than nothing" privacy with bloom filters: To what extent? In *International Conference on Privacy in Statistical Databases*, pages 348–363. Springer, 2012.
- [5] Microsoft. How to use random hardware addresses in windows. [Online]. URL: <https://support.microsoft.com/en-us/windows/how-to-use-random-hardware-addresses-in-windows-ac58de34-35fc-31ff-c650-823>
- [6] Odysseas Papapetrou, Wolf Siberski, and Wolfgang Nejdl. Cardinality estimation and dynamic length adaptation for bloom filters. *Distributed and Parallel Databases*, 28:119–156, 2010.
- [7] Ori Rottenstreich and Isaac Keslassy. The bloom paradox: When not to use a bloom filter. *IEEE/ACM Transactions on Networking*, 23(3):703–716, 2014.
- [8] SecDev. [Online]. URL: <https://scapy.net/>.
- [9] David Stritzl. Privacy-preserving matching using bloom filters: an analysis and an encrypted variant, April 2019.
- [10] Zero Tier. [Online]. URL: <https://www.zerotier.com/>.
- [11] European Union. Gdpr. [Online]. URL: <https://gdpr.eu/tag/gdpr/>.
- [12] Wikipedia. Coap - constrained application protocol. [Online]. URL: https://en.wikipedia.org/wiki/Constrained_Application_Protocol.