TÉCNICO LISBOA

SureSpace: orchestrating IoT devices to certify location

João Maria Marques Tiago

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Prof. Miguel Filipe Leitão Pardal

Examination Committee

Chairperson: Prof. José Carlos Martins Delgado
Supervisor: Prof. Miguel Filipe Leitão Pardal
Member of the Committee: Prof. João Carlos Serrenho Dias Pereira

January 2021

À Avó Alice, pelo farol e porto de abrigo que sempre foi e será.

Ao Professor Miguel Pardal, pela confiança que sempre manifestou em mim, pelo conhecimento que, com motivação, me transmitiu durante o mestrado, por todas as ideias, sugestões e correções que contribuíram para a qualidade final do documento, e pela inesgotável compreensão ao longo de tempos menos bons.

A toda a Família, pelo apoio constante, pelo amor pleno, pela paz que sempre me dá.

A todos os Amigos, e, em particular ao Gonçalo, pelo alicerce emocional permanente.

Acknowledgments

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2019 (INESC-ID) and through project with reference PTDC/CCI-COM/31440/2017 (SureThing).

Resumo

As aplicações móveis que recorrem à localização do dispositivo são cada vez mais populares e úteis. Contudo, à medida que mais serviços dependem da localização, crescem as preocupações com utilizadores que reportam falsas localizações para ganhar vantagens indevidas. Uma forma de prevenir a falsificação da localização é confiar em sistemas de certificação de localização. Por exemplo, o SureThing recorre ao Wi-Fi ou a transmissores Bluetooth e a testemunhas *ad hoc* para oferecer a possibilidade de um utilizador provar a sua localização num determinado momento e num determinado local. Esta abordagem pode ser alargada a espaços como edifícios inteligentes, geridos por plataformas como o DS2OS. Neste trabalho, propomos, implementamos e avaliamos o SureSpace, um novo sistema que combina certificação da localização com a gestão de espaços inteligentes para verificar a localização de utilizadores dentro de um edifício inteligente. O novo sistema recorre a infraestrutura existente que utiliza como testemunha, e, para alcançar maior segurança, introduz diversidade nos dispositivos utilizados e nas técnicas de prova, com o objetivo de impedir a falsificação da localização. O sistema foi avaliado em condições de laboratório, e mostrou ser eficaz utilizando sinais luminosos e de som.

Palavras-chave: Certificação de Localização, Contextualização de Aplicações, Segurança de Dispositivos Móveis, Espaços Inteligentes, Internet das Coisas

Abstract

Location-aware mobile applications are increasingly popular and useful. However, as more services rely on location, there are concerns that users may misreport their location to gain undue advantages. One way to prevent such location spoofing is to rely on location certification systems. For example, SureThing uses Wi-Fi or Bluetooth beacons, and ad-hoc witnesses to allow a user to make proof of location at a specific time and place. This approach can be extended to smart spaces, such as smart buildings, managed by platforms like DS2OS. In this work, we propose, implement, and evaluate SureSpace, a new system that combines location certification with smart space management, to verify the location of users inside a smart building. The new system relies on existing infrastructure to act as witnesses, while introducing a varied combination of devices and proof techniques to achieve security by diversity in the proofs, and thwart spoofing attacks. The system was evaluated in laboratory conditions, and was shown to be effective using light and audio signals.

Keywords: Location Certification, Context-Awareness, Mobile Security, Smart Spaces, Internet of Things

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mobile devices, like smartphones and tablets, or even more computationally constrained devices, such as smartwatches, have become an essential part of people's everyday lives [HHFM17]. The mobile commerce grew very rapidly, as vendors launched new devices with a rich variety of technical features [HC16], turning mobile devices into sophisticated machines with powerful capabilities. As a result of technological innovations, mobile applications, or *apps*, designed to run on mobile devices to provide specific functionality, have appeared from the convergence of media, information technology, and Internet [PD18]. There are several categories of mobile applications, ranging from social media and games, to utility and productivity, and their popularity has led to the emergence of an *app economy*, with corresponding developments in supply and demand of mobile applications [BK11].

The use of *location* in mobile applications to deliver location based services (LBS) became popular in the last decade. These applications deliver information tailored to the location and context of the mobile device and the user [HGK+18]. Different groups of LBS have been considered, from location based social networks, that add a location dimension to existing social networks, to location based gaming, that maps real-world environments into a virtual world. However, mobile devices may lack a trustworthy mechanism to collect location in a verifiable way, and users may feel temped to misreport their location to gain undue access to a restricted resource provided by a LBS [SW09].

To prevent location spoofing, location certification mechanisms can be deployed to prove that a user is at a specific location (either geographical or logical), at a specific time, while offering strong security properties. SureThing [FP18] is a recent proof of location framework built with Java on the Android platform that uses location measurements collected with GPS, Wi-Fi, and Bluetooth Low Energy (BLE) to certify location. SureThing makes use of *witness* devices to verify and attest to the presence of users in crowded physical spaces with diverse devices. In its

current version, it does not take advantage of existing infrastructure available at a location.

Also in the last decade, the Internet of Things (IoT) emerged as a network of interconnected smart devices [PP+16] that exchange context resources transparently for management and decision-making purposes, depending upon characteristics like network availability or coverage area [BMBB14]. Depending on their type, smart devices use different protocols to communicate, expose and process ambient resources in different ways, and have varying information-processing capabilities [VBCMV+12]. IoT environments are, then, strongly dynamic, but smart device heterogeneity poses serious interoperability issues, preventing the development of complex pervasive scenarios where smart devices work together to deliver valuable services in a smart space. To solve this problem, Pahl proposed the Distributed Smart Space Orchestration System (DS2OS) [Pah14], a pervasive framework designed to facilitate smart space management. DS2OS allows discovering and configuring smart devices regardless of their specifics, enabling the development of complex orchestration schemes that can be perceived as smart space services.

In some smart environments, mobile devices can interact with IoT devices embedded into the infrastructure for collaborative user-centered, and scenario-based services. Securing these interactions is a critical concern, if one is to allow a mobile device to execute actions on an IoT infrastructure. However, relying on the assumption that the user's profile suffices to determine what they are authorized to do is not enough. Following a certain security policy, some actions may require the physical presence of the device in a specific location, and, for that reason, context attributes, like user identity, time, and location, are the basis for good security decisions when granting access to infrastructure resources [ACD+06].

With this work, we introduce SureSpace, an extended version of the original architecture of SureThing, designed to engage in smart environments in a secure way. To generate a location proof, *beacons*, which are on-site smart devices, broadcast unique signals meant to be captured by the prover using *witnesses* in the prover device. If the captured signal matches the original signal to a certain threshold, the location proof is deemed valid. Different approaches and techniques are supported for signal processing and matching. To discover, configure, and control beacons inside the smart space, we use DS2OS [Pah14], a smart space management framework. It bundles a beacon discovery mechanism to account for the high dynamism of smart environments, allowing beacons to be added or removed in runtime.

To evaluate SureSpace, we designed a small-scale smart infrastructure by deploying a LED, a light sensor and a speaker, and developed an Android client application to test the prototype. The setup, using inexpensive equipment, is a realistic and plausible version of a real-world one.

1.1 Objectives

We proposed SureSpace, a location certification mechanism designed for smart environments. It leverages smart devices as beacons that broadcast preconfigured signals, and witnesses that capture these signals, to later attest to the presence of the user. SureSpace is based on a previous location certification system, and resorts to a smart space orchestration framework to discover, configure, and control beacons. To the best of our knowledge, this is a novel approach to collect proof of location in smart environments. We developed a prototype of SureSpace, and evaluated it in laboratory conditions. We also evaluated its resistance to attacks, based on the established attacker model.

1.2 Outline

The remainder of the document is structured as follows. Chapter 2 provides an in-depth look into smart environments, explores different ways to represent location, discusses the most prominent techniques and technologies for indoor localization, and addresses indoor localization systems, proof of location systems, and smart space management. Chapter 3 provides insight on the architecture of SureSpace. Chapter 4 covers the most relevant details about the implementation of the SureSpace prototype. Chapter 5 presents the experimental setup used to evaluate the prototype, describes the evaluation criteria, and discusses the evaluation results in a controlled scenario, and under attack. Chapter 6 summarizes important points of our work by identifying expected contributions, and suggests relevant features left as future work.

Chapter 2

Background & Related Work

This Chapter explores the key concepts of the three fundamental domains that compose the theoretical framework of SureSpace: *smart environment*, *indoor localization*, and *proof of location*. Section 2.1 introduces the core concepts of the Internet of Things, and discusses its layered architecture proposed by [PP+16]. Section 2.2 addresses the pertinence of representing locations with coarser granularity in SureSpace, and provides examples of geocode systems. Section 2.3 provides insight into abstract approaches to indoor localization. Respectively, Sections 2.4 and 2.5 cover the most common techniques and technologies used for indoor localization, and Section 2.6 explores indoor localization systems that stand out by their disruptive approaches to localization. Section 2.7 motivates for the need of certifying location of a mobile device, and discusses some proof of location systems in real-world scenarios. Finally, Section 2.8 stresses the importance of management in smart spaces, and provides a real-world example of a system designed to achieve smart space orchestration.

2.1 Internet of Things

At a lower level, the *Internet of Things* (IoT) is a network of physical objects, of all types and sizes, ranging from *smart sensors* and home appliances, to cameras and medical instruments [PP+16]. These devices are interconnected through one or more common channels, and share information based on stipulated protocols, in order to provide a wealth of intelligence for planning, management, and decision making [BMBB14]. Interconnectivity, heterogeneity, dynamic changes, and scalability characterize the IoT environment.

IoT is an admixture of multiple and diverse technologies and communication standards, aiming to provide end-to-end connectivity. Current research is primarily focused on long-range machine-to-machine connection, but existing short- and medium-range solutions will not be

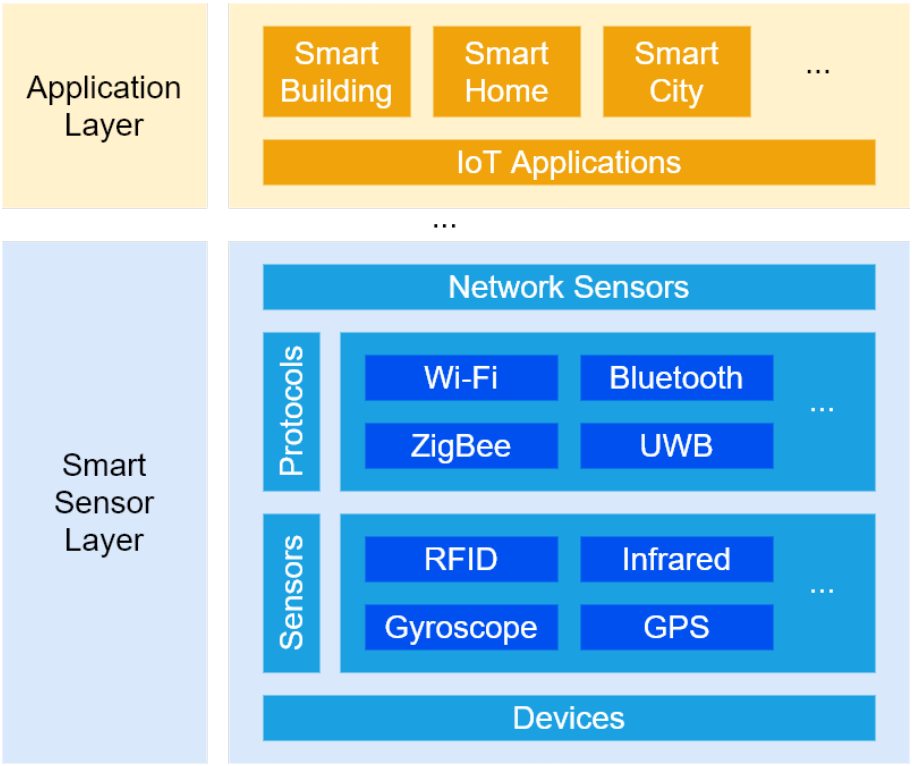


Figure 2.1: A partial view of the IoT architecture, adapted from [PP+16].

wiped off the map in the near future. This is because long-range technologies grant a high coverage with low power communication solution, whilst IoT applications in local level require high data rate. Thus, in the near future, IoT devices might turn into double-interface devices: one for short-range communication, and another one for long-range communication [ZGL19]. Figure 2.1 covers two of the four layers of the architecture of the IoT paradigm, according to [PP+16].

Following a bottom-up approach, the *Smart device/sensor layer* is made up of *smart devices* integrated with *sensors*, that connect the physical world to the digital world. These sensors measure physical properties (like temperature or pressure), and convert them into a signal ready to be shared with other devices. For that purpose, most sensors are connected to a network, either wireless or wired, through sensor gateways (that use protocols like Wi-Fi or Bluetooth).

The *Application layer* covers *IoT applications*, named after the collaborative cooperation of smart devices to reach common goals. Some applications are more viable and appropriate than others, depending upon characteristics of the *smart environment*, such as network availability, coverage, scale, user involvement, or impact [GBMP13]. A *smart home*, for example, is a kind of smart environment expected to be deployed in a small area, with few active users. Most smart devices will be connected to the local network, to offer services such as device control, weather forecast, and intrusion detection mechanisms [PP+16]. *Smart buildings* are a type of smart envi-

ronments too, and seek to integrate and account for intelligence and control, with adaptability, in order to meet energy and efficiency, longevity, and comfort and satisfaction [BMBB14].

Figure 2.1 deliberately omits two middle layers, the *Network/communication layer* and the *Application support layer* (bottom to top). Respectively, these layers handle (1) the transfer of massive volumes of data in a way to support low latency and high bandwidth, and (2) the pre-processing of that data through analytics and security controls before reaching the top layer. Although important, they lose relevance in the context of our research because the problems they cover are already handled (refer to Section 2.8).

2.2 Location representation

Geocodes represent geographic locations on Earth, where each location is assigned a unique identifier, used to distinguish between entities. Usually, geocodes are short and human-readable. In SureSpace, we use geocodes to locate both the prover and the orchestrator within the building. Our problem domain allows locations to be represented with coarser granularity, because the focus is on determining if the prover is within the boundaries of the room, and not necessarily at a specific location within that area.

This section covers the two geocode systems supported by SureSpace: the Open Location Code (refer to Subsection 2.2.1), and What3words (refer to Subsection 2.2.2).

2.2.1 Open Location Code

The Open Location Code (OLC) geocode system [plu] encodes locations into *plus codes* that represent squares on the Earth's surface. The longer the code, the smaller the square is, and, in its full length of 11 characters, a plus code represents a 3×3 m square. For example, `8CCGPVP5+GMW` describes the reception hall of a building in Lisbon. Shortening the code to `8CCGPVP5+GM`, a larger square containing the reception hall is drawn. This feature is useful to represent larger rooms with a single plus code.

2.2.2 What3words

The What3words (W3W) geocode system [wha] follows a similar approach: each 3×3 m square on the Earth's surface is assigned a unique 3-word address that does not change over time. The grid spacing is immutable, meaning larger rooms might require multiple codes to be fully covered. For reference, `///spotted.muddle.folds` is the W3W code that represents the previously described reception hall.

Table 2.1: Abstract approaches to device localization.

Name	Description
Device based localization (DBL)	The user resorts to a subset of Reference Nodes (RN) to compute their relative location. Primarily used for navigation.
Monitor based localization (MBL)	A subset of the RN is passively obtaining the position of the user connected to the node. Primarily used for tracking.
Proximity Detection (PD)	Distance is estimated between user and a Point of Interest (PI), like a beacon.

2.3 Indoor localization overview

As smartphones and wearable devices with wireless communication capabilities proliferate in a wide-scale, the location of such devices becomes a valuable context. Their technological capabilities and diversity range from low-cost sensors to sophisticated smart devices, capable of collecting data and interacting with the external environment, enabling the development of location-based applications and services. Device localization has a broad range of applications, from the health sector to building management.

Although most IoT technologies have not been designed with localization capabilities, some applications might require pervasive and smooth indoor localization of both mobile and static devices. Existing short- and medium-range communication technologies can estimate relative indoor positions, with varying degrees of accuracy, with respect to some reference point. The absolute location of the device, however, remains unknown, unless the global location of the reference points is known a priori. When it comes to long-range technologies, and because the infrastructure relies on static access points, an accurate global location of the device is usually made possible. However, the overall accuracy of the estimation is still very low, specially concerning indoor environments, where the GPS signal is virtually non-existent.

A close collaboration between short- and medium-, and long-range solutions is required to diminish current market demand for high coverage and high power communications, brought by diverse localization requirements of IoT devices. Consequently, and before introducing the different localization techniques and technologies, it becomes necessary to differentiate between the abstract approaches to device localization. Table 2.1 considers different approaches covered in [ZGL19].

Existing user localization solutions have shifted from network- to user-centric approaches, Location Based Services (LBS), from which both service providers and end users can benefit a lot.

2.4 Indoor localization techniques

This Section introduces common indoor localization techniques: Received Signal Strength Indicator (refer to Subsection 2.4.1), Angle of Arrival (refer to Subsection 2.4.2), Time of Flight (refer to Subsection 2.4.3), Time Difference of Arrival (refer to Subsection 2.4.4), and Fingerprinting (refer to Subsection 2.4.5). Subsection 2.4.6 is a short summary of the different techniques.

2.4.1 Received Signal Strength Indicator

One of the simplest and most widely deployed approaches for indoor localization is based on the *Received Signal Strength* (RSS), i.e. the actual signal strength power the receiver gets, usually measured in decibel-milliwatts (dBm). With the help of signal propagation models, it is possible to turn the RSS value into an absolute and physical distance, because the signal strength power at a reference point is known beforehand. The greater the RSS value, the closer receiver and transmitter are from one another. However, RSS is not a localization technique, but rather an approach to tackle the problem. The actual technique is the RSS indicator (RSSI), a relative measurement of the RSS, with no standard units (commonly defined by chip vendors).

RSS based localization can be used in both DBL and MBL cases. In the DBL case, the RSS at the device is used to compute the absolute distance between the device and, at least, three RN (like Wi-Fi access points). Upon that, trigonometry rules are applied to determine the device location relative to the above-mentioned RN. Thus, such approach requires trilateration. In the MBL case, the RSS value received at the RN is used to compute the device location. This is a centralized approach, because further communication between all participant RN is required for a distributed collection and processing of RSS values.

Although simple and cost effective, RSS based approaches suffer from poor localization accuracy. This is mainly due to signal attenuation, multipath fading, and indoor noise [ZGL19]. Filtering mechanisms can be applied to increase accuracy — although reaching higher levels of accuracy remains infeasible without bringing into play complex and computationally expensive algorithms.

2.4.2 Angle of Arrival

Angle of Arrival (AoA) based localization approaches require multiple antennas at the receiver (an antennae array) that work as a single antenna. By calculating and exploiting the time difference of arrival at each of the individual antennae, it is possible to determine the angle at which the signal reaches the sensor and, thus, the device location [ZGL19]. For an n-dimension environment, n antennae at the receiver are required to compute their location. Therefore, in

a real 3D setting, only 3 antennae are required. Also, AoA delivers good estimation accuracy when the distance between the transmitter and the receiver is relatively small. However, the farther the transmitter and receiver are, the more deteriorated the accuracy gets — a slight error in the angle calculation becomes a large error in the location estimation. To deliver the same level of accuracy, this approach requires more complex algorithms and careful calibration, when compared to RSS based approaches. Moreover, because this technique is affected by multipath effects, the line of sight required for efficiently working might be difficult to obtain.

2.4.3 Time of Flight

Also known as Time of Arrival (ToA), *Time of Flight* (ToF) capitalizes on the electromagnetic signal propagation time to determine the distance between transmitter and receiver. By multiplying the measured ToF value by the speed of light in a vacuum, the physical distance between transmitter and receiver can be gauged, according to Equation (2.1)

$$D_{ij} = (t_j - t_i) \times c \tag{2.1}$$

where D_{ij} is the distance between transmitter i and receiver j, t_i is the time at which transmitter i sends the message, t_j is the time at which receiver j receives it, and c is the speed of light in a vacuum. Furthermore, $t_j = t_i + t_p$, where t_p is the time taken by the signal to traverse from transmitter i to receiver j.

Similar to RSS based-systems (if one considers the ToF value an RSSI value), this technique requires a static architecture of, at least, three known access points, that act as receivers. In a first phase, resorting to Equation (2.1), the absolute distance between the receiver and each transmitter is computed. Upon that, basic geometry is applied to calculate the location of the device with respect to the involved access points.

For this technique to be correctly deployed, receivers' and transmitters' clocks must be synchronized and, depending on the requirements imposed by the underlying protocol, timestamps might be required to be transmitted along with the signal. The estimation accuracy of such technique is deeply coupled to both the signal bandwidth and the sampling rate. If the sampling rate is too low, the signal might arrive between the sampled intervals, and thus not be captured. To assuage such problem, frequency domains super-resolution techniques are put in place to extract the ToF value with high resolution [ZGL19]. However, as stated earlier, indoor environments suffer from multipath fading, requiring larger bandwidth to counterbalance possible signal losses. Nevertheless, although both improvements add to a better estimation of the ToF value, they cannot eliminate significant errors when no direct line of sight can be drawn [ZGL19].

Existing objects in indoor sites may deflect the original signal, which then propagates through a longer path, tampering with the expected ToF values because of longer measured times.

2.4.4 Time Difference of Arrival

The *Time Difference of Arrival* (TDoA) technique exploits the difference in signals propagation times emitted by different transmitters and measured at the receiver [ZGL19]. Unlike the ToF technique, which relies on the absolute signal propagation time to compute the absolute distance, this technique resorts to TDoA measurements (which are delta-Ts) to determine the distance between each transmitter, according to Equation (2.2)

$$L_{D(i,j)} = c \times T_{D(i,j)} \tag{2.2}$$

where $L_{D(i,j)}$ is the absolute distance between transmitters i and j, c is the speed of light in a vacuum, and $T_{D(i,j)}$ is the difference in signals propagation times measured at the receiver. Upon that calculation, an hyperboloid, depicting all possible receiver locations, is drawn for each pair of transmitters, given by Equation (2.3)

$$\begin{aligned} L_{D(i,j)} = &\sqrt{(X_i - x)^2 + (Y_i - y)^2 + (Z_i - z)^2} \\ &- \sqrt{(X_j - x)^2 + (Y_j - y)^2 + (Z_j - z)^2} \end{aligned} \tag{2.3}$$

where (X_i, Y_i, Z_i) are the coordinates of transmitter i, and (x, y, z) are all possible coordinates of the receiver. At least three transmitters (and, thus, at least three hyperboloids) are required to determine the exact location of the receiver — the intersection of all hyperboloids. The resultant system of equations of hyperboloids can be solved either through linear regression or by using Taylor-series expansion to linearize the equation [ZGL19]. Similar to the ToF, the overall accuracy of the TDoA technique depends on the signal bandwidth, sampling rate (at the receiver), and the existence of a direct line of sight between transmitters and receiver. Unlike the ToF, the TDoA does not require clock synchronization between transmitters and receiver, but rather strict synchronization between the transmitters.

2.4.5 Fingerprinting

Also known as scene analysis, *Fingerprinting* is a technique that encompasses two different phases. The first one is the offline phase, and it requires an environmental survey to gather fingerprints of the site where the system is going to be deployed in — usually, RSSI measurements. In the second phase, and in real-time, online measurements are collected and compared against the stored measurements to assess the user location. To understand the extent to which both

measurements match, and to map the device location on a grid, different approaches have been taken, and are discussed in [ZGL19].

The first and oldest approach relies on probabilistic, and embodies so-called *Probabilistic Methods*. Using mathematical models, it is possible to compute the likelihood of the device being in a given location provided the features measured at a given location. This estimation is not continuous, but rather discrete, and, theoretically, the smaller the distance between each offline measurement point, the more granular the estimation is, because the measurement grid is denser. However, in such cases, the signal difference between two neighbor points might become much smaller than the standard signal indoor signal variation, drastically decreasing the accuracy of the location estimation. Therefore, there is a significant tradeoff to consider between the offline grid density and the probability of a successful estimation. Also, one should bear in mind that indoor environments are susceptible to change over time, requiring new site fingerprinting.

Probabilistic Methods paved the way for *Artificial Neural Networks* (ANN), used to classify and forecast indoor scenarios. In an initial phase, the ANN is trained using both the measurement point's absolute coordinates, and the RSSI values read at that location from the offline phase. Once it is trained, the ANN can be used to estimate the device location taking the online RSSI measurements as input.

Another approach finds its roots in the process of comparing location points that are exclusively in proximity to the point we wish to locate. *k-Nearest Neighbor (kNN)* algorithms rely on RSSI online measurements to determine the k-nearest matches (based on offline RSSI measurements previously surveyed), using root-mean-square error (RMSE). To obtain the estimated device location per se, the k-nearest matches are averaged (possibly weighting each match, if distances are adopted as weights).

2.4.6 Summary

In this section, we covered the different state-of-the-art indoor localization techniques. Table 2.2 provides a comparison between the different techniques.

2.5 Indoor localization technologies

This Section introduces common indoor localization technologies: Wi-Fi (refer to Subsection 2.5.1), Bluetooth (refer to Subsection 2.5.2), Radio Frequency Identification (refer to Subsection 2.5.3), Visible Light (refer to Subsection 2.5.4), Acoustic Signal (refer to Subsection 2.5.5), and Ultrasound (refer to Subsection 2.5.6). Subsection 2.5.7 is a short summary of the different technologies.

Table 2.2: Comparison of localization techniques, adapted from [ZGL19].

Technique	Advantages	Disadvantages
RSSI	Easy to implement and cost-efficient.	Low localization accuracy due to environmental noise, might require fingerprinting prior to use in a real context.
AoA	Delivers high localization accuracy, and does not require fingerprinting.	Requires complex hardware and algorithms, and the performance deteriorates with increase in distance between transmitter and receiver.
ToF	High localization accuracy without requiring fingerprinting.	Requires synchronized clocks between the receiver and the multiple transmitters, and, perhaps, time stamped messages.
TDoA	No need for clock synchronization between receiver and transmitters, nor fingerprinting.	Requires synchronized clocks between the multiple transmitters, and larger bandwidth.
Fingerprinting	Easy to deploy on site.	Minor changes in the indoor environment require new site surveying.

2.5.1 Wi-Fi

Under the official name of *IEEE 802.11 standard*, Wi-Fi aims to provide networking capabilities, and access to the Internet to devices in public, commercial or private environments [ZGL19]. Although initial reception ranges were low, the need for connection of more and more mobile devices in space regions increasingly smaller led to the development of *IEEE 802.11ah*, with a range of about 1 km, and centered on IoT devices. With the current proliferation of mobile devices with networking capabilities (especially devices with an attached Wi-Fi adapter that users tend to keep enabled for long periods of time), this technology plays a pivotal role in indoor localization [ZGL19]. The numerous and almost-omnipresent Wi-Fi access points have been used as reference nodes for most localization techniques, reducing the overall cost of implementing them, given that no additional infrastructure is needed.

Although existing Wi-Fi networks are being used for localization purposes, they were originally deployed with the intent to provide high data throughput and network coverage to devices in the vicinity of its access points. Thus, more efficient and more robust algorithms are further required, to ensure greater localization accuracy, while reducing possible noise interference in the communication channel induced by existing devices. Any of the aforementioned localization techniques (RSSI, ToF and AoA, or any combination of them, i.e. hybrid methods) can be adopted to provide Wi-Fi based localization services [ZGL19].

2.5.2 Bluetooth

Bluetooth, or, officially, *IEEE 802.15.1 standard*, is a physical and MAC layers specification designed to connect wireless devices within a certain personal space [ZGL19]. Versions 4.x of the protocol (starting with version 4.0, known as *Bluetooth Low Energy*, or BLE) aim to lower power consumption, while offering lower latencies, and high nominal data rates (up to 8 Mbps) [ZGL19]. Version 5 of the protocol (the latest version) has new features focused on new IoT technology. It ensures a communication bandwidth that allows data transfer among multiple smart devices at nominal rates that go up to 16 Mbps [CPTT18], while offering larger operation ranges (up to 200 m outdoors). Although most techniques, such as RSSI, AoA and ToF, can be deployed with an infrastructure composed by BLE beacons, the vast majority of BLE based localization solutions rely on RSS based inputs [ZGL19]. These techniques are less complex, but the indoor localization accuracy is quite limited. BLE is, thus, a good candidate for indoor localization using beacons, but context aware proximity based services, such as *iBeacons* (by Apple Inc.) or *Eddystone* (by Google Inc.), are proprietary systems.

2.5.3 Radio Frequency Identification

It is possible to transfer and store data over an electromagnetic transmission from a *Radio Frequency Identification Device* (RFID) tag (the transmitter), and a radio frequency compatible circuit (the receiver). A priori, both transmitter and receiver must agree on a specific radio frequency and protocol to communicate, so that the receiver can read the data sent by the RFID tag. Two different RFID systems are mentioned in the literature: active RFID, and passive RFID [ZGL19]. These approaches differ greatly in communication range, frequency range, power source, overall size, and cost.

Active RFIDs operate in the ultra high and microwave frequency ranges, covering areas of hundreds of meters between transmitter and receiver. Periodically, active RFID tags transmit their ID, whilst connected to a local power source. Although such systems have been used for object tracking at a "low" cost, they cannot achieve sub-meter accuracy, and are not a portable solution.

On the other hand, passive RFIDs operate with the power provided by the reader signal, which they reflect to send back information. They deliver good connectivity when both transmitter and receiver are close in the vicinity. Passive RFID tags are smaller, cheaper and work in many frequency ranges, from low frequency to microwave, going through high and ultra-high frequencies. Currently, these tags are replacing bar-codes, but remain unsuitable for indoor localization due to the low range.

2.5.4 Visible light

Light Emitting Diodes (or LEDs) can be used to modulate and emit visible light between 400 and 800THz, opening room for high-speed data transfer using light, the so-called *Visible Light* Communication (VLC). LEDs act as light beacons, transmitting the signal, which is collected and interpreted at the receiver's sensor, measuring the position and direction of the LEDs. Upon that, and using the AoA technique, it is possible to accurately estimate the relative position of the receiver [ZGL19]. Currently, this technology has a major drawback: it requires an unobstructed line of sight between the LED and the sensor. Nevertheless, and given that LEDs are used in several rooms in the house, this technology may become widely used.

2.5.5 Acoustic signal

The vast majority of smartphones carry built-in microphone sensors. Such ubiquity can be leveraged to estimate the relative device location with respect to some preset sound emitting reference nodes (special speakers, for instance). The *Acoustic Signal* technology involves the transmission of modulated acoustic signals, containing timestamps or other relevant information, collected by the device's microphone sensor for ToF estimation [ZGL19]. However, the smartphone's microphone sensors have some limitations (like the low sampling rate, and the anti-aliasing filter), posing some difficulties for indoor localization — accurate estimations are only feasible if acoustic signals with frequencies under 20KHz (audible band) are used. To achieve such behaviour, the transmission power must be low enough not to create noise sound pollution (i.e. humans should not be able to hear it), and signal processing to improve the signal received by the sensor is further required. Notwithstanding, acoustic signal is not widely used, because it requires high sampling rate (which might drain the device battery too rapidly), and additional infrastructure (as reference nodes need to be placed on the spot).

2.5.6 Ultrasound

The *Ultrasound* based localization technology relies on ToF measurements of ultrasound signals (not audible, with frequencies above 20KHz), and the sound velocity to compute the absolute distance between a transmitter and the receiver. The idea is similar to the calculation from a time interval, described in Subsection 2.4.3. It has been proven to provide indoor localization with centimeter-level accuracy, supporting the tracking of multiple mobile nodes at the same time within the same vicinity, with high energy efficiency, and virtually non-existent signal leakage between rooms [ZGL19]. Usually, a previous radio frequency pulse is broadcast, to ensure the necessary synchronization, because receiver and transmitter might not be clock-synchronised

when the ultrasound transmission begins. Despite all the apparent advantages, the sound velocity, unlike radio frequency signals, varies drastically when room temperature, and humidity change [ZGL19]. To remedy such drawback, temperature sensors are normally deployed along with the ultrasound emitters on the spot, so the correct adjustments can be performed. Still, the overall accuracy of ultrasound based localization is low in the presence of high levels of environmental noise — depending on the intensity of the noise, complex signal processing algorithms might be applied to filter it out.

2.5.7 Summary

In this section, we covered the different state-of-the-art indoor localization technologies. The maximum range is indicative of the area the technology might cover, and the power consumption is relevant because, as explained later, low energy consumption is one of the paramount priorities to run a wide-scale IoT infrastructure at lower costs. Table 2.3 provides a comparison between the different technologies.

2.6 Indoor localization systems

Section 2.3 introduced different abstract approaches for indoor localization. Over time, indoor localization systems have shifted from network- to device-centric approaches, and DBL approaches have gained popularity. Network-centric systems rely on the wide distribution and availability of some infrastructure elements, increasing the deployment costs. On the other hand, DBL approaches do not rely on the adjacent infrastructure, since the device itself connects to a subset of RN to determine its location. In line with this trend, disruptive approaches to DBL have been proposed in the literature, with systems that leverage sensors embedded into the device to achieve good localization accuracy.

This Section presents representative examples of some of these systems, based on acoustics (refer to Subsection 2.6.1), BLE (refer to Subsection 2.6.2), light and magnetic sensors (refer to Subsection 2.6.3), or sound fingerprinting (refer to Subsection 2.6.4). SureSpace is not an indoor localization system, but understanding the feasibility of these approaches, and their design implications is of valuable help.

2.6.1 Acoustics based

Some indoor localization systems are coarse-grained, with room- or, at best, meter-level accuracy. Although such guarantees might suffice in some settings, like outdoor navigation, they will not meet the requirements in the centimeter-level realm, where accuracy is paramount. To surmount

Table 2.3: Comparison of localization technologies, adapted from [ZGL19].

Technology	**Maximum range**[a]	**Power consumption**	**Advantages**	**Disadvantages**
Wi-Fi	35 m	Moderate	Does not require extra devices, widely deployed and high accuracy.	Requires complex algorithms processing and is prone to ambient noise.
Bluetooth	40 m[b]	Low	Low energy consumption and wide range.	Delivers low accuracy and is prone to ambient noise.
RFID	200 m	Low	Low energy consumption and wide range.	Low localization accuracy.
Visible Light	1.4 km	Relatively high	Available at a wide-scale and delivers high accuracy.	The effective range is affected by objects in the line of sight path, and comparatively consumes more energy.
Acoustic Signal	Few meters	Low-moderate	Provides high accuracy.	Requires extra sound emitters and is severely affected by noise pollution.
Ultrasound	A few tens of meters	Low-moderate	Supports tracking multiple mobile nodes at the same time.	High dependencies on the sensor placement.

[a] Unless stated otherwise, the reported values are for indoor environments, and taken from [ZGL19].
[b] According to [CPTT18].

such problematic, Liu *et al.* have proposed Guoguo [LLL13], an indoor localization ecosystem that uses acoustic signals to estimate the device location. This disruptive technology is used due to recent research concluding that leveraging ubiquitous microphone sensors (which most smartphones have) introduces a new approach of ranging by using the audible band acoustic signal. Besides ubiquitous, these sensors are inexpensive and deliver high localization accuracy due to the low transmission speed of acoustic signals. Nevertheless, Liu *et al.* faced some difficulties, because of the limited bandwidth of microphones, the strong attenuation of aerial acoustic signal, and the various interferences in the audible band, which all translate to short operation distance, low update rate, and sound pollution [LLL13].

The proposed ecosystem uses on-site deployed reference nodes that act as acoustic beacons (inaudible to the human ear), and a smartphone application responsible for localization processing, thus performing passive sensing, i.e. collecting and analysing the transmitted signal passively. However, instead of emitting simple "Beep" signals, and in order to support multiple smartphone users, Liu *et al.* designed transmission waveform, wide-band modulation, and one-way synchronization, and ranging schemes. They also propose additional mechanisms to improve the ecosystem performance with regard to coverage, update rate, and sound pollution, like a fine-grained adaptive time of arrival estimation approach that exploits the details of the beacon signal, and performs non-line-of-sight identification and mitigation [LLL13]. Although a localization accuracy ranging from 6 to 25 *cm* can be achieved using Guoguo, the acknowledged drawbacks deteriorate the performance to values that make it unsuitable for a ubiquitous localization system [ZGL19].

2.6.2 BLE based

One of the main radio technologies emerging in the scenario of user-centric LBS is the BLE. Given its low power communication, as discussed in Subsection 2.5.2, it is ideal for indoor environments, where the received RSS value has significant fluctuations, turning such technology into the perfect candidate for provisioning contextual aware services. Following up with such idea, Zafari *et al.* propose a novel approach that leverages Apple's *iBeacons*, intended for proximity, to build a LBS [ZP15], and further evaluate the effectiveness of the model with regard to the accuracy of indoor positioning. *iBeacons* allow applications running on smartphones, either Android or iOS, to receive such signals.

According to the specifications of the *iBeacon* protocol, every beacon is permanently broadcasting advertisement packets, carrying a universal identifier (with the purpose of differentiating between multiple organizations), along with other values used to separate beacons belonging to

one organization according to their physical location (for instance, a shop section). A smartphone near the beacon is able to listen to the signal being transmitted, and using the RSSI technique (refer to Subsection 2.4.1) it is capable of estimating the device location with high accuracy. Upon that, the application running on the smartphone contacts a server to retrieve a context-aware entity (like a discount coupon), based on its location.

Zafari *et al.* leverage this mechanism to prototype a high-accuracy system for indoor or GPS-constrained environments through *particle filtering*. This is one of the theoretic approaches to follow in order to increase the tracking accuracy in the environment, proven to perform better than other filters [ZP15]. The main contribution of their work lies on the particle filtering parameter tuning, which enhances localization accuracy up to 0.97 m, but also on the main conclusions they draw with respect to the positioning of the beacons. According to their experimental results, increasing the number of beacons will only increase the accuracy until the area is saturated with beacons [ZP15]. Also, overloading the space with beacons might affect the accuracy adversely, because beacons interfere among each other. Furthermore, beacons must be placed in the ceiling, to avoid obstacles.

2.6.3 DeepML for indoor localization with smartphone magnetic and light sensors

Wang *et al.* propose a novel approach to indoor localization using a smartphone's magnetic and light sensors [WYM18]. Without requiring additional infrastructure, this approach exploits bimodal magnetic field and ambient light data with a deep learning approach. The geomagnetic field and light intensity, at a given location, are highly stable and robust over time (i.e. an ubiquitous signature of the site), and, at many locations, are complementary to each other. Because local anomalies to both data are known, combining such data allows to increase the size of input data, improving location diversity, recognition performance and, therefore, the overall accuracy of the indoor localization process. To train bimodal data, a deep long-short term memory (LSTM) network, a popular recurrent neural network (RNN), was used. Compared to fingerprinting based methods, the deep LSTM network does not require the creation of a database: instead, it only needs one group of weights trained for all training locations.

Similar to other approaches, the overall process comprises three steps: data preprocessing and training (performed offline), and location estimation (performed online). In the first one, the site is surveyed to obtain real-time readings from both sensors (the sampling rate of the magnetic field sensor is adjusted, to ensure synchronization with the ambient light sensor). The offline training is the novel approached proposed in [WYM18]. A fully connected layer was implemented

for location features extraction from the sensors' raw data. Upon that, a two-layered deep LSTM network is used for training optimal weights, and a classifier is used to train the output data. In the third and last step, the bimodal image representing the unknown location is fed to the trained deep LSTM network to estimate the device location.

Their experimental study demonstrates the feasibility of the process, and the effectiveness of the system under two representative indoor environments (a laboratory scenario, cluttered with tables, chairs, and computers, and a corridor scenario).

2.6.4 RoomSense

RoomSense [RSA+13] discusses a novel approach to indoor room-level positioning, using an *active sound fingerprinting*[1] approach. The system was deployed to an Android smartphone, using its incorporated speaker and microphone, requiring no additional hardware.

A given room is characterized by a set of *room acoustic* and *common audio* features, useful to fingerprint a given position inside the room. Not all features suit the purpose of fingerprinting, because some of them lead to low performance, or are inefficient. Because it outperformed all the other features individually, the Mel-Frequency Cepstral Coefficients feature was chosen, according to their experimental evaluation. Proactively, the smartphone running RoomSense measures the *impulse response*[2], in particular the Maximum Length Sequence, in a given position. Later, the collected data for that location is processed (feature extraction and selection are performed, for instance), and classified (using dynamically generated room models — there is a training phase involved).

Training impulse responses for multiple positions and smartphone orientations inside a given room allowed for a room-level localization accuracy of 98.2 %, according to [RSA+13]. Nevertheless, the recognition accuracy is dependent on the density of training positions per room — in order to reach a room localization accuracy of 80 %, at least one position every 18 m^2 should be trained. Furthermore, they prove their system is robust against ambient noise (a localization accuracy above 80 % is still possible with a *signal-to-noise-ratio* above 30 dB). Also, RoomSense, when compared to other similar systems, is faster, requiring less than 1 s to go through all the required steps to output a location estimation.

2.6.5 Summary

In this section, we covered a few disruptive systems that follow a DBL approach for indoor localization. Without extra infrastructure, they leverage smartphone sensors to achieve high

[1] Process of emitting a sound chirp and measuring the impulse responses.
[2] The impulse response is a response of a dynamical system to a Dirac input impulse.

Table 2.4: Comparison between indoor localization systems.

Approach	Accuracy	Advantages	Disadvantages
Acoustic signals	Up to 6 *cm*	High localization accuracy.	Short operation distance, and low update rate. Prone to interferences in the audible band (background noise).
BLE	Up to 97 *cm*	Energy efficient (since BLE offers low power communication). Great for indoor environments, where RSS values have significant fluctuations.	Significant operational delay. Additional costs since it requires the deployment of iBeacons.
Magnetic field and light intensity	Up to 50 *cm*	No extra infrastructure. Geomagnetic field and light intensity are highly stable and robust over time.	Requires site fingerpinting.
Sound fingerprinting	Room-level	No extra infrastructure. Low latency, and robust against ambient noise.	Requires site fingerpinting.

accuracy localization. Where possible, we stated the main conclusions and design implications of each system, by looking at the conducted experimental evaluation. Table 2.4 provides a comparison between the different systems.

2.7 Proof of location systems

Since access points started having embedded geographical information, it has been observed that people carry Bluetooth- or Wi-Fi–enabled devices on them, some capable of running location-aware applications. Such applications use location to deliver services tailored to the user's context. For instance, a hypothetical credit card with enhanced security can be programmed to only work in the vicinity of some locations chosen by the user (say a city or region). In this scenario, the system would have to verify and validate the user's location prior to approving any transaction. Under some circumstances, users may feel tempted to mock their location to gain undue access to restricted resources [SW09]. Taking on the previous example, an attacker that obtains physical access to a credit card has a financial motivation to lie about their real location and get unlimited access to the privileges of the credit card's owner. The problematic is aggravated by the fact that most devices lack trustworthy mechanisms to prove their location (such as expensive trusted platform modules to make GPS data unforgeable). Ever since, the need for *location proofs*, that certify a user to a specific location and time, has emerged.

A typical location proof has five fields: an issuer, a recipient, a timestamp, a location,

and a digital signature [SW09]. Identities (both issuer's and recipient's) are usually proved by public keys, securely certified by a trusted third-party. Desirably, a proof of location mechanism should offer security properties like integrity (to ensure the proof cannot be tampered with), non-transferability (to bind the proof exactly to one user), and privacy (to prevent the disclosure of the user's location, current or past). Varying degrees of proof verification are possible, depending on the considered physical attacks to the system. For instance, users may deliberately pass their mobile devices to other users, pretending to be somewhere else — spoofing attack.

This Section covers the location certification processes of different location certification systems. In Subsection 2.7.1, we provide insight into the abstract mechanism. Then, we present three systems: APPLAUS (refer to Subsection 2.7.2), Crepuscolo (refer to Subsection 2.7.3), and SureThing (refer to Subsection 2.7.4).

2.7.1 Workflow overview

All three location certification systems covered in this Section share similar architectures and workflows. To initiate the process, the user wishing to prove their location, the *prover*, estimates their location using whatever mechanism the system provides. Then, the prover collects evidence from other users in the vicinity, known as *witnesses*, that attest to their presence. Finally, the system verifies the evidence collected by the prover to determine the legitimacy of the claim.

2.7.2 APPLAUS

Zhu *et al.* propose APPLAUS [ZC11], a user-centric location privacy model in which co-located Bluetooth-enabled mobile devices mutually generate location proofs, and update to a location proof server. Mobile nodes in the same vicinity communicate with each other to exchange location proofs using Bluetooth.

In the message flow, one of the nodes, playing the role of the *prover*, wishes to prove its location, and broadcasts a location proof request to nearby nodes. That request is received by *witnesses*, that, upon approval, generate a location proof and send it back to the prover. To keep track of history records of the location proofs, a *location proof server* is needed. That server is untrusted, and prover nodes submit their location proofs directly to it. To protect mobile nodes privacy, APPLAUS considers the existence of a *certification authority*, run by an independent and trusted third-party, that manages the credentials (the identity) for the nodes, serving as a bridge between the verifier and the location proof server. Prior to joining the network, and for each mobile node, the certification authority preloads a set of public/private key pairs — public keys are known as *pseudonyms*. Therefore, and even if an attacker obtains complete network

coverage and tracks nodes throughout the entire network, the certificate authority will still be the only party who knows the mapping between real identity and pseudonyms. Finally, the system considers the existence of a *verifier*, i.e. a third party who is authorized to verify a prover's location.

The proposed privacy model gives mobile nodes the capability of assessing their privacy level, and the capacity to decide if a proof request must be accepted, based on the current level of privacy. According to their experimental results, APPLAUS delivers location proofs effectively while preserving nodes privacy at the same time.

2.7.3 Crepuscolo

Canlar *et al.* propose Crepuscolo [CCCDP13], a more robust collusion resistant and privacy preserving location verification system. Canlar *et al.* introduce the concept of *neighbor-based solutions*, in which location proofs are acquired from neighboring mobile devices (whilst in *infrastructure-based solutions* location proofs are acquired from infrastructure elements, like access points, for instance). Crepuscolo (and, thus, APPLAUS) are examples of neighbor-based solutions.

In the message flow, Crepuscolo adopts the same entities already discussed in APPLAUS (see Subsection 2.7.2). They have similar names and play the same role. However, APPLAUS only protects against simple collusion attacks (using a reputation system), not addressing the capability of the attacker to perform a *wormhole attack*, a specific type of collusion attacks. The attacker records a packet at one location inside the network, tunnels the packet to a colluding party at another location, and replays the packet there [CCCDP13]. To tackle this type of attack, Crepuscolo adds a new entity, the *token provider*, which generates a token for each location proof. A token is a piece of information which endorses the location proofs acquired from witnesses. They can only be created with the physical involvement of the prover, and can not be reconstructed at a different location. Therefore, location proofs are endorsed by a token, and the combination *proof of location* + *token* proves that a certain mobile node was at a certain location, at a certain time, providing the necessary resilience against more complex collusion attacks. To preserve privacy, Crepuscolo adopts the same mechanism already discussed in APPLAUS, *pseudonyms*.

According to their experiments, the proposed system is able to detect up to 90 % of the collusion attacks, using few token providers in a large area. They further compare their system to APPLAUS with respect to location verification, and protection against wormhole attacks.

2.7.4 SureThing

SureThing [FP18] is a location certification system for Android devices that lets users prove their location. Its design and architecture are influenced by APPLAUS (refer to Subsection 2.7.2), and Crepuscolo (refer to Subsection 2.7.3), to the extent that they share some core components.

Figure 2.2 offers a simplified view over the location proof mechanism provided by SureThing, with special emphasis on the different steps of the process. The prover starts by estimating their location using different *location estimation technologies*, such as Geo Proof (geographic location obtained from GPS or ANLP[3]), Wi-Fi Proof (via Wi-Fi fingerprinting), and Beacon Proof (using BLE beacons, refer to Subsection 2.5.2). After clearance to start the location proof, the prover collects evidence from witnesses in their vicinity that attest to their presence. To validate the proof, SureThing challenges the original claim by verifying the evidence collected by the prover.

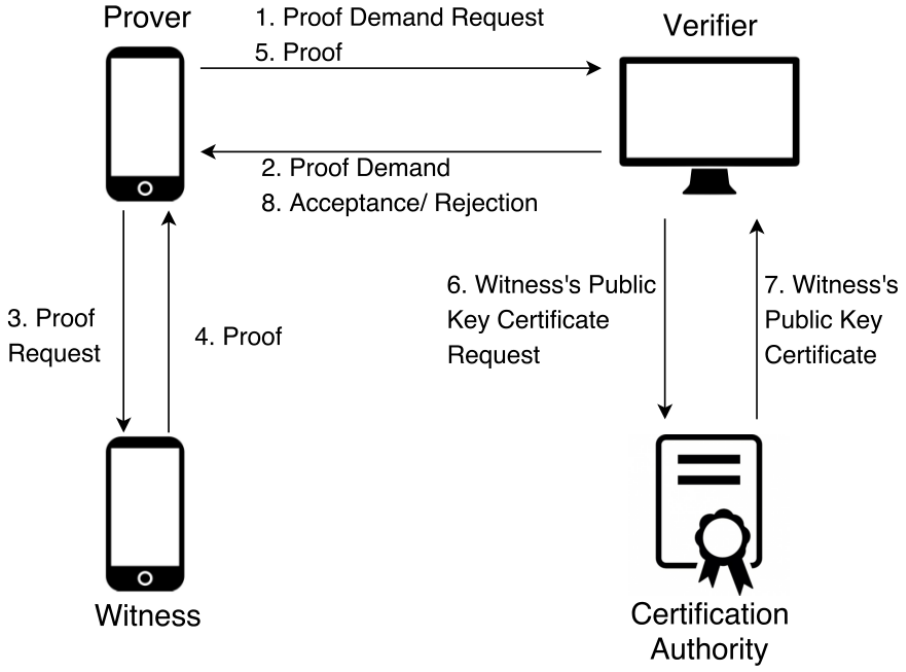


Figure 2.2: SureThing components and communication flows.

At a more abstract level, we can consider *evidence* collected by the prover to be some sort of *signal*, that is unique to the location at a specific time. In Figure 2.2, that signal is information sent by witnesses in the vicinity to confirm the user's presence, but that signal can be extended to be a "natural" signal, like the radiation or noise levels. Naturally, the prover collects the signal with errors and keeps the evidence for later verification. When the prover wants to make proof of location and time, they are challenged by the verifier. The verifier can ask for the signal or some of its features, and compare them with a template (like observations reported by witnesses).

[3]The Android Network Location Provider, that uses both cell tower and Wi-Fi to determine the device location.

Table 2.5: Comparison between proof of location systems.

System	Advantages	Disadvantages
APPLAUS	Resorts to pseudonyms to protect mobile nodes privacy.	Only protects against simple collusion attacks.
Crepuscolo	Addresses wormhole attacks, offering higher protection against collusion attacks.	The mechanism is more complex because tokens are used to endorse location proofs.
SureThing	Offers multiple location estimation techniques to locate the user.	Only protects against simple collusion attacks.

SureThing uses a *certification authority*, that is trustful and binds identities to public keys. To guarantee *collusion avoidance*, SureThing uses *witness redundancy*, and *decay* mechanisms — with promising results in crowded locations. For the majority of places, Wi-Fi Proof (with 10 readings) was the technique that ensured the highest accuracy. Also, such technique is the one with the most regular readings regarding time spent — turning it into the best option for location estimation.

Since the original SureThing paper, there have been other applications implemented with the approach, some with ad-hoc witnesses, others with fixed witnesses, and also some with beacons that transmit pseudo-random signal sequences. There is also an ongoing work that is providing witness and user *privacy protection* through the use of different privacy mechanisms [CP20]. Until this work, SureThing lacked a way to leverage *multiple* signals and the smart space orchestration capabilities, required for a robust and diverse proof of location process.

2.7.5 Summary

This Section motivated for the need of proof of location systems, and explored real-world examples of existing systems. For each system, we described the main components of its architecture, the attacks it can handle, and its privacy model. Table 2.5 provides a comparison between the different systems.

2.8 Smart space management with DS2OS

The IoT presents a scenario with numerous heterogeneous computational actors that greatly increase processing capacity [VBCMV+12]. These actors exchange context resources transparently through one or more communication channels, offering the possibility for a new paradigm which would improve people's lives. However, interoperability issues between the different actors, with regard to their capabilities and offered resources, are one of the major obstacles that lie between

what IoT *is* and what it *could* be. This heterogeneity comprises the communication protocols adopted by each actor, the different ways they expose ambient resources, and how they transform raw data into high-level information, ready to be consumed by upper-layer services. Semantic models might be the best way to describe actors and the surrounding ambient in pervasive scenarios [VBCMV+12], such as smart spaces, where smart devices play the role of the described actors. To cope with the lack of a standardized approach to smart space management, we looked for a "smart space orchestrator".

In an effort to improve the orchestration capabilities of a smart space, Pahl introduced the Distributed Smart Space Orchestration System (DS2OS) [Pah14], a service-oriented framework focused on the development of services for smart spaces. Services are logical processes that deliver functionality in a smart space, usually grouped in two categories: *adaptation services*, that provide an interface to a smart device, and *orchestration services*, that implement the logic behind pervasive scenarios. In practice, each smart device requires an adaptation service, and smart devices are coordinated by an orchestration service. Regardless of their category, services have unique identifiers in DS2OS, conveniently named after the nature of the service (e.g. the adaptation service for a temperature sensor *could* go by `temperatureadaptationservice`).

Smart devices — be them sensors, that acquire data from the world, or actuators, that change the state of a physical space — play the starring role in smart spaces, as they deliver valuable data (e.g. light or temperature conditions), and perform useful work (e.g. turn a light on or turn heating on). These capabilities can be leveraged to implement real-world pervasive scenarios where smart devices work together towards a common goal (e.g. turn heating on if room temperature drops below some threshold). The coordinated management of smart devices located in a smart space is known as smart space orchestration, and becomes possible if smart devices can be interconnected to share data over a network. Without further support, however, full-fledged orchestration of smart spaces is commonly deemed infeasible, because interconnection problems between heterogeneous smart devices, and the lack of an "orchestration brain", block the way for the development of more complex orchestration schemes.

Smart devices produce and consume pieces of unstructured information, known as *context*. To become manageable, context is shaped into *context models*, that represent entities in a structured way. Context models have *context nodes* (i.e. attributes) to describe properties of the entity they are linked to. To determine its type, each context node has a `type` attribute, which is, in fact, a context model itself, usually simpler. To understand this better, consider a context model of a temperature sensor, of type `/sensor/temperature`. The context model should include, at least, two properties: `isOn`, a `/boolean` for the operational status of the sensor, and `value`, a

`/number` used to represent the read temperature. The `temperatureadaptationservice` would, then, include a context node of type `/sensor/temperature` to represent the sensor. In other words, all properties of the sensor's context model would be implicitly included in that context node.

Each service has its own context model, and inter-service communication is achieved through the manipulation of context models. Simply put, a context model is a persisted blackboard: some services write on it, and other services read from it. To prevent unauthorized operations, context nodes have read and write permissions. Following the example, all services should be able to change the value of `isOn`, so that the sensor can be switched on. However, `value` must be read-only to all services but the `temperatureadaptationservice`, authorized to update the temperature value. In this light, context models are interfaces for services, a sort of *service contract* that (1) specifies which attributes can be read and/or written, via `get` and `set` operations, respectively, and (2) by whom, according to an *access control policy*. For instance, if an orchestration service wishes to get the current temperature, it would have to call `set('isOn', true)` on the context model of the `temperatureadaptationservice` (to enable the sensor), and then `get('value')` to get the actual temperature.

Context models are stored in a distributed system over a peer-to-peer network, known as the Virtual State Layer (VSL). Peers of the network are called Knowledge Agents (KAs), and each agent persists a subset of the context models. To be granted access to the distributed knowledge, services register to a KA of their choice, that becomes responsible for the respective context models. From that moment, these context models become available to other services, even if connected to a different agent.

One important feature of DS2SOS is that services can subscribe to specific context nodes to receive a notification when the node changes. This feature is useful, for instance, when a service wants to take different actions depending on the new value. Another key feature of DS2OS is dynamic service discoverability [AL19, PL19], that allows adaptation services to be discovered by different criteria: type of smart device (e.g. `temperatureadaptationservice`), or type of attribute (e.g. `/boolean`).

Figure 2.3 summarizes key ideas of DS2OS. Services (in yellow) connect to exactly one Knowledge Agent (in green), that persists their context models on a *context repository* managed by a *context manager*. Context models (more specifically, their context nodes) can be manipulated through `get` and `set` operations, and services can `subscribe` to certain context nodes to be notified of changes (the `notify` callback) — accesses are mediated by the agent and validated against an access control policy. Smart devices (in blue), both sensors and actuators, are

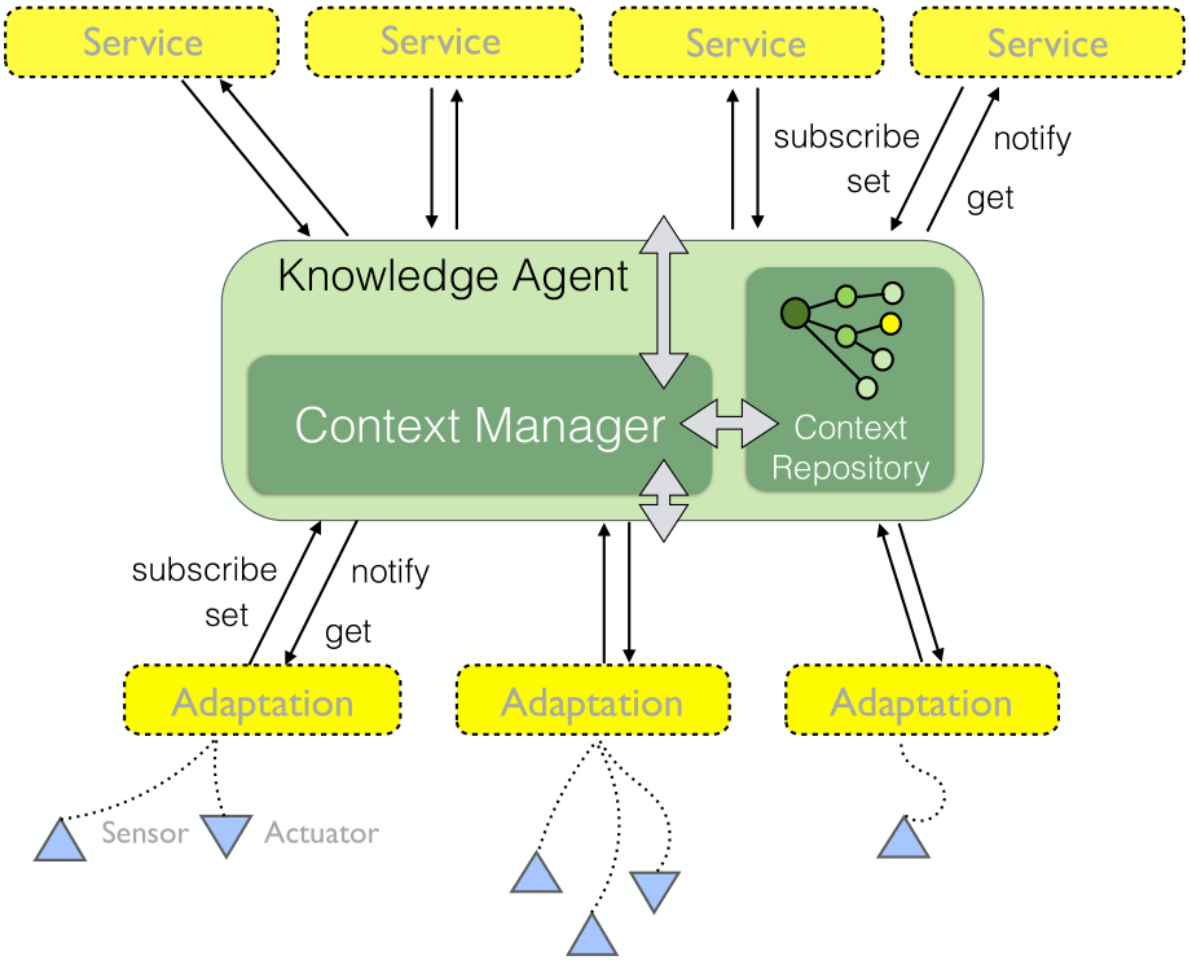


Figure 2.3: Context management architecture of the VSL, adapted from [Pah14].

controlled via (adaptation) services.

2.9 Summary

In this Chapter, we started by covering the IoT, with a brief overview of its layered architecture, and an explanation of the most relevant terms that relate to the topic. We motivated for the use of geocodes to represent locations, providing examples of real-world geocode systems. We covered the most prominent indoor localization techniques and technologies, based on the state-of-the-art systems addressed in [ZGL19]. We motivated for the need of proof of location systems in real-world scenarios. Finally, we addressed the importance of management in smart spaces, later relevant in the architecture we propose.

Chapter 3

SureSpace design

SureSpace is a location certification mechanism designed for smart environments. Its architecture results from the integration of the SureThing and DS2OS frameworks, covered in Chapter 2, for location certification and smart space management, respectively. On the one hand, the SureThing domain encompasses all components necessary for the proof of location (Certificate Authority, Witness, Prover, and Verifier), enriching SureSpace with an assortment of technologies for location certification. On the other hand, the DS2OS domain brings the possibility to configure and orchestrate smart devices in a smart space, as well as the possibility to dynamically discover them in run-time (introducing components like Knowledge Agents, and the Orchestrator).

As an example, consider a meeting room in a smart office building where SureSpace has been deployed. Employees, prior to a meeting, can be required to use their mobile phones to generate a location proof, to prove that they were at that specific room, at a specific time. Unlike other location certification systems, that rely on other users in the vicinity to prove location, SureSpace uses mobile devices that interact with the smart and trusted infrastructure in the room to generate the location proof.

Necessarily, communication between mobile devices and the infrastructure is wireless, exposing the system to external threats. In general terms, we can describe a SureSpace attacker as any rouge user that deliberately engages in a dishonest behavior with the intent of proving their false presence. At a lower level, we consider that an attacker is capable of:

1. Passively listen to all communications (eavesdrop);
2. Impersonate a user or a device;
3. Tamper with the data exchanged between source and destination.

To secure SureSpace, communication between components should use existing secure channel implementations (like TLS), to provide confidentiality, authenticity (integrity and data source

authentication), and freshness (to prevent replay attacks). Whenever a custom communication channel is considered, we assume that there is no interference, and no remote readings due to the limited range. These are the key points we took into account when designing SureSpace, that we introduce next.

Section 3.1 offers an overview of SureSpace. Section 3.2 details each component of the architecture, explaining its purpose, and how it relates to other components. Section 3.3 explores the different stages of the certification process, required to generate and verify a location proof in SureSpace. Section 3.4 is a brief summary of the Chapter.

3.1 Overview

SureSpace applies the concept of *signal*, introduced by SureThing (refer to Subsection 2.7.4), to smart environments. To prove location, it explores interactions between the *prover device* (the mobile device used by the prover to engage with SureSpace) and *beacons* (a subset of smart devices in the trusted infrastructure). In a first stage, beacons broadcast signals meant to be collected, with errors, by the prover device. Multiple factors contribute to signal degradation, leading to a "degraded" version of the original signal. In SureSpace, a location proof is a collection of (degraded) signals captured by the prover device. In a second stage, signals are compared: if the degraded version of the signal matches its original version to a certain extent, it is considered legitimate. A location proof is accepted if enough signals are legitimate. The specific threshold to legitimate a signal is application-dependent, and must be defined in each verifier implementation, as described in Section 4.4.

For the sake of simplicity, the terms *smart space* and *smart environment* are used interchangeably to designate a physical space where a cluster of smart devices has been deployed. Moreover, consider that smart spaces can be divided into multiple *rooms*.

3.2 Architecture

In this section, we introduce all components of the SureSpace domain. To help visualize the logical structure of the solution, we use Figure 3.1, the deployment diagram of SureSpace. In the diagram, green is used for artifacts, blue for nodes, yellow for devices, and orange to distinguish the smart space infrastructure from the other nodes. Components are presented following the expected interaction sequence in a proof of location scenario.

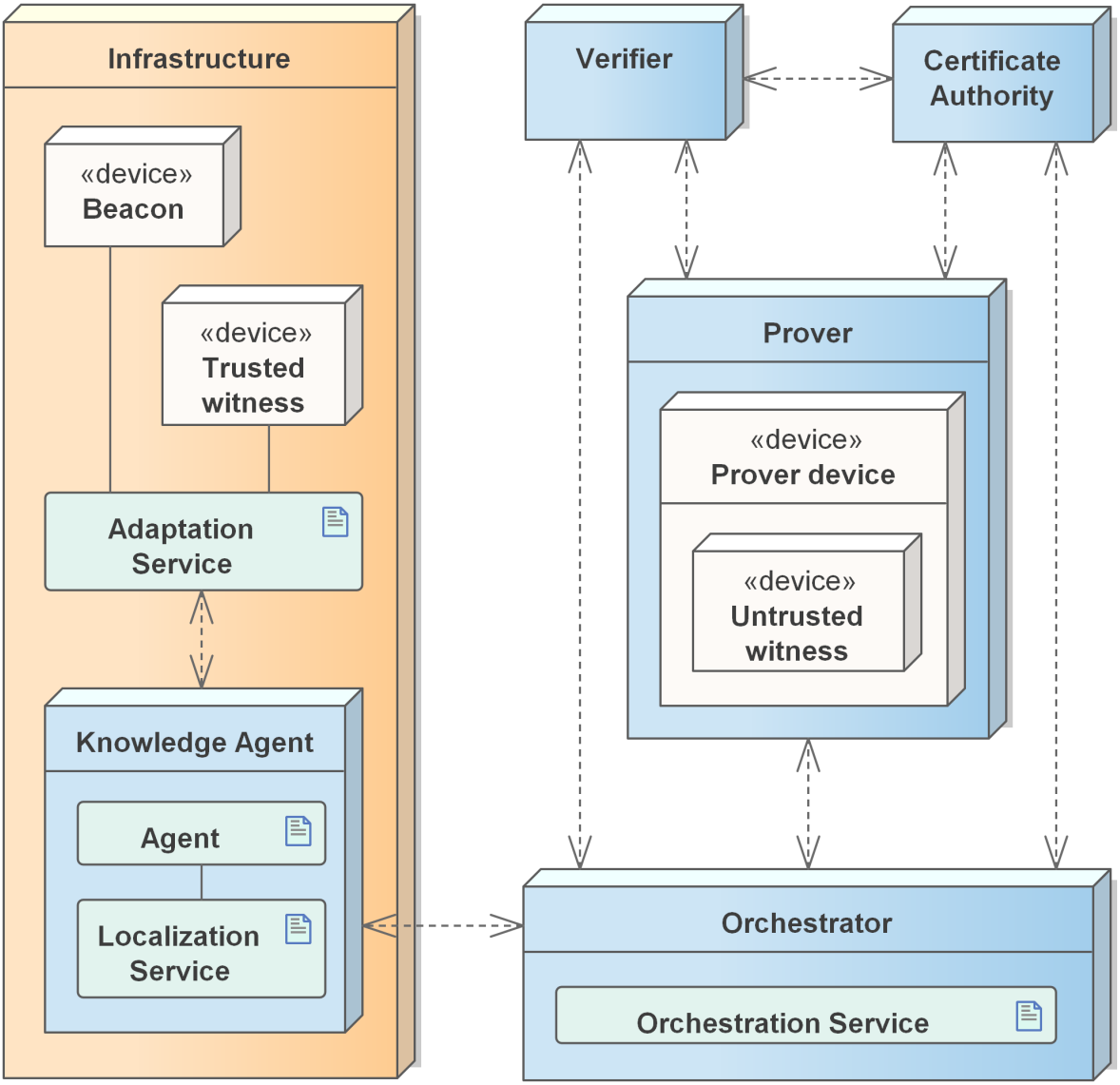


Figure 3.1: SureSpace deployment diagram.

3.2.1 Prover

The prover is a SureSpace user that engages with the system in order to prove their location. The prover device is the device used by the prover during all interactions with the system.

3.2.2 Certificate Authority

The Certificate Authority (CA) is the long-term identity provider of all active entities of the system, similar to CAs in use on the Internet for website certification.

Each entity generates a public/private key pair. The private key is known only to the entity, and is kept safe and secure on their side. The public key is used to generate a certificate signing request in order to apply for a public key certificate (mandatory step equivalent to *entity registration*). If the request is approved by the CA, a public key certificate is issued and assigned to the requester entity.

SureSpace is able to handle the registration of new entities at runtime by supporting *dynamic instantiation*. For example, a new prover can join SureSpace at any given time to prove their location, or a new orchestrator can be deployed, configured, and immediately set to work. To support this, entities are grouped by type, and certificates assigned to entities of the same type are nested under the same intermediate CA. Taking on the previous example, certificates assigned to provers belong under the *Prover CA*, and certificates assigned to orchestrators belong under the *Orchestrator CA*. Of course, intermediate CAs belong under the *Root CA* of SureSpace, and, by the way certificate chains work, so do all other certificates. Figure 3.2 provides visual aid to understand how certificates are organized in the hierarchy of trust adopted in SureSpace.

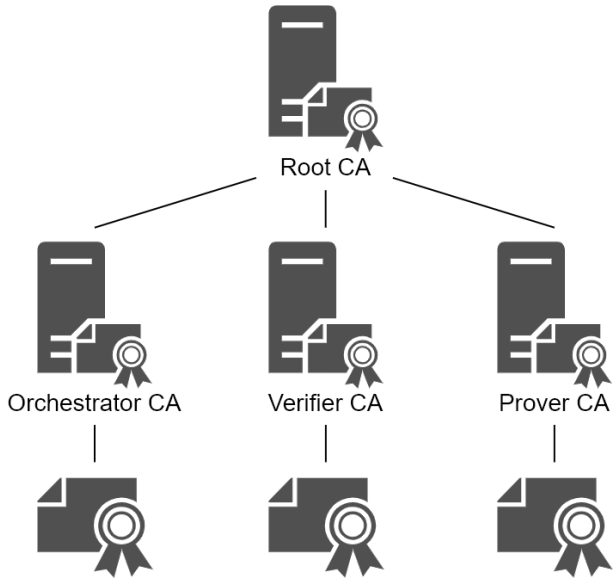


Figure 3.2: Hierarchy of trust adopted in SureSpace.

For communication purposes, entities are required to have *unique identifiers* within the SureSpace domain. These identifiers are hierarchical and generated based on the entity's certificate chain, that starts with the Root CA and ends with the certificate assigned to the entity (refer to Subsection 4.1.2 for a comprehensive example).

3.2.3 Orchestrator

From the prover's perspective, an orchestrator is the "entry point" to SureSpace because it is the component they first reach out to when engaging with the system. An orchestrator is the "mastermind" of any proof of location, responsible for coordinating the process at the highest level, and dispatching communication flows to prevent malicious actions coming from unauthorized parties. To that extent, it implements diverse logical subprocesses that include, for example, the dynamic discovery of new orchestrated rooms, the dynamic discovery of new beacons and their orchestration, and the delivery of accurate information about a specific location proof. Orchestrators do not persist any information: all relevant activity (either communication- or proof-related) is stored in volatile memory.

Depending on network congestion generated by provers trying to prove their location, more orchestrators can be instantiated to increase the throughput. At a structural level, SureSpace was designed to accommodate one orchestrator per smart space. A smart building, for instance, can be divided into multiple smart spaces (e.g. one per floor), thus requiring more than orchestrator. To prove their location, the prover can connect to any of the available orchestrators. By default, to save transmission time, the prover will connect to the closest orchestrator. For example, each smart space can display a QR code that encodes the identifier of the orchestrator to contact.

Orchestrators run an *orchestration service* to communicate with the adaptation services of the orchestrated beacons (since the distributed knowledge is only available through the VSL). Since the orchestration service is simply used to encapsulate low-level calls to the VSL into higher-level functions, used by the orchestrator to implement its functionality, its context model has no context nodes.

3.2.4 Knowledge Agent

A Knowledge Agent (KA) is a node in the distributed knowledge network of DS2OS (refer to Section 2.8). More specifically, KAs are context repositories that persist relevant information used by an orchestrator.

To deliver room-level orchestration, each orchestrated room has its own KA, that behaves like a *proxy* to the room (meaning that no rooms share the same KA). Agents hold information

about their geographical location, so that the orchestrator can discover them by location when looking for new orchestrated rooms. For that purpose, each KA runs a *localization service* (a type of DS2OS service), that can be queried to retrieve the agent's location. Figure 3.3 depicts a simplified context model of the localization service. `location` is the string representation of the agent location using a plus code (refer to Section 2.2).

```
<model type="/complex/service">
    <location type="/basic/text"/> <!-- OLC representation of the agent location -->
</model>
```

Figure 3.3: Simplified context model of the localization service.

Additionally, new beacons must be discoverable within a specific orchestrated room. Since beacons register themselves to the closest KA in their vicinity (refer to Subsection 3.2.5), new beacons are easily accounted for. Thus, a KA aggregates beacons logically by orchestrated room in such a way that the logical boundaries of the area are ultimately drawn by the geographical dispersion of the connected beacons, regardless of the real physical boundaries of the room.

During a proof of location, an orchestrator will need information about engaging beacons (and respective trusted witnesses), like the value of specific attributes. For that reason, alongside with relevant metadata, context models of the registered beacons are part of the persisted information in KAs. That information becomes available to other KAs, since they are all nodes in the same distributed knowledge network.

3.2.5 Adaptation Service

Beacons require a *proxy* to become discoverable and controllable by SureSpace (refer to Subsection 3.2.6). To achieve that, each beacon has an *adaptation service* (a type of DS2OS service), that connects the beacon (and the trusted witness) to SureSpace. Each adaptation service has a different context model, required for inter-service communication (refer to Section 2.8), that includes the represented entities (beacon and trusted witness) as context nodes.

As an example, consider an adaptation service (`lightadaptationservice`) that controls a smart bulb (the beacon), and a light sensor (the trusted witness). These two devices require appropriate context models that mirror their properties. Figure 3.4 shows the representation of a simplified context model of a smart bulb. `isOn` is a boolean used to control the beacon (if set to `true`, the beacon is activated).

```
<model type="/complex/smartbulb">
    <isOn type="/basic/boolean"/> <!-- boolean status of the beacon -->
</model>
```

Figure 3.4: Simplified context model of a smart bulb.

Analogously, Figure 3.5 shows the representation of a simplified context model of a light sensor. `samplingRate` controls how many readings the sensor does per time unit, and `intensity` is the value read by the sensor.

```
<model type="/complex/lightsensor">
    <samplingRate type="/basic/number"/> <!-- how many readings per time unit -->
    <intensity type="/basic/number"/> <!-- light intensity (any range) -->
</model>
```

Figure 3.5: Simplified context model of a light sensor.

Figure 3.6 shows the representation of a simplified context model of the described adaptation service. It has two context nodes, `beacon` and `witness`, that represent the two entities controlled by the service. Note that each context node sets the `type` attribute to match the corresponding context model. This way, all properties of the device are implicitly included in the context node (refer to Section 2.8).

```
<model type="/complex/lightadaptationservice">
    <beacon type="/complex/smartbulb"/>
    <witness type="/complex/lightsensor"/>
</model>
```

Figure 3.6: Simplified context model of a light adaptation service.

To summarize, an adaptation service is a source of information. By looking at it, it becomes evident which devices are being controlled, and what configuration parameters are available for each of them based on the properties of the respective context model. With this information, the adaptation service implements beacon-specific functionality.

Adaptation services have a locking mechanism that prevents concurrent use by multiple orchestrators. Before a beacon is activated for a proof of location, an orchestrator tries to acquire the lock over the respective adaptation service. If the attempt fails, the proof of location is aborted. Only when the same orchestrator releases the lock, which should happen in the end of the proof of location, may other orchestrators activate the beacon. In practical terms, this

means that beacons are locked for a specific proof of location, and cannot be used at the same by other orchestrators.

Ideally, a signal broadcast by a beacon is available within a certain radius of action, and must not go beyond the physical boundaries of the orchestrated room. Thus, it makes sense that the beacon is bound to a specific geographical location, where it is available. Since KAs are already aware of their location (refer to Subsection 3.2.4), adaptation services must register themselves to the KA responsible for their immediate vicinities.

3.2.6 Beacon and Witnesses

A *beacon* is a smart device embedded into the trusted infrastructure ready to be used in a proof of location (e.g. a smart bulb). Beacons are usually actuators (they change the state of a physical space), and exhibit special behavior when compared to other smart devices. By default, SureSpace is not aware of existing beacons by themselves, since they may not be directly connected to the system. Thus, each beacon requires a *proxy* to become visible to, and controllable by the system. That proxy is an *adaptation service*, that represents and controls exactly one beacon (refer to Subsection 3.2.5). During the proof of location, each beacon generates and broadcasts exactly one *signal* meant to be captured by, at least, one corresponding *witness* in the prover device. A signal is something produced by a beacon that can be received by a corresponding witness. For instance, visible light, emitted by a smart bulb (the beacon), and acknowledged by a light sensor (the witness), could be used as a signal. In that case, it would be something *physical*, conveyed from beacon to witness. On the other hand, a random QR code on a display (the beacon) could also be used as a signal, and it could be acknowledged by reading the word with a camera (the witness). Thus, the definition is left open to avoid narrowing down the possibilities to a small set of conventional signals, paving the way for out-of-the-box ideas. Theoretically, any equipment can be used as a witness, provided it is able to receive signals from a specific beacon. Witnesses in the prover device are *untrusted*, because they are not part of the trusted infrastructure. Ergo, in SureSpace, we redefine the concept of a *proof of location* as a series of signals, issued by a subset of beacons, that are received by, at least, one corresponding witness in the prover device.

A signal is generated based on a set of *quirky properties*, that feed a deterministic signal generator. If these properties are disclosed, the original signal can be easily replicated. Naturally, signals have different characteristics/features, and are susceptible to degradation induced by multiple factors during their transmission. Moreover, witnesses have limited capabilities, and might not be able to acknowledge all the characteristics of the signal, but only a subset of

them. Thus, what the witness receives is not the original signal, but a *degraded* representation of some of its characteristics. In some cases, part of the characteristics of the signal can be successfully derived from the analysis and/or processing of the degraded representation. To the derived information we call *proof ambient*, because it represents the witness' knowledge of the original signal. To maximize the accuracy of the proof ambient, diverse corresponding untrusted witnesses, capable of acknowledging different characteristics of the signal, can be used simultaneously to derive more information.

To determine the legitimacy of the location proof, we measure the accuracy of the proof ambient by quantifying similarity between the original signal, that was transmitted by the beacon, and the degraded representation, that was received by the witness. To do that, we require a *trusted representation* of the signal from a *trusted witness*, embedded into the reliable infrastructure, capable of acknowledging the same set of characteristics, to ensure the two representations are compatible. In some cases, usually when the infrastructure is not open for modification, trusted witnesses can be *virtual*. Without the need for real devices, virtual witnesses mimic the behavior of receiving the signal through emulation to deliver the same functionality. Alongside with a beacon, corresponding trusted witnesses, be them virtual or not, are represented and controlled by the same adaptation service (refer to Subsection 3.2.5).

Witnesses may have limited storage capacity (if any at all). That means that, in most cases, the collected information is ephemeral, and gets lost if not stored in time. Different approaches can be used to cope with this situation, usually by sending the information to a different entity, that stores the information.

3.2.7 Verifier

Regarding the certificate processes chain, the verifier is the last entity to engage in SureSpace. It measures the reliability of a location proof by assessing the legitimacy of the proof ambient derived by the prover (refer to Subsection 3.2.6).

Depending on the supported beacons, the verifier must implement adequate criteria to ensure that signal representations can be properly compared. For that reason, there is no pre-determined assessment criteria for evaluating a location proof, since the definition of signal itself remains abstract enough to encompass a variety of beacons. Moreover, verification can take into account application-specific criteria.

Regardless of the implementation details, the verifier must output a boolean value that represents the assessment result. If `true`, the location proof is accepted. In the end of the evaluation, more information can be yield by the verifier (like a confidence interval).

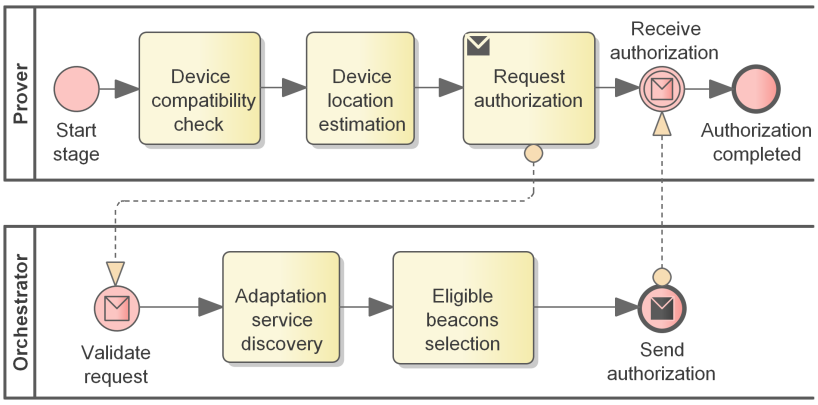


Figure 3.7: Process diagram for the pre-authorization stage.

3.3 Location certification process

This Section details the certification process required to generate and verify a location proof in SureSpace. The process encompasses three stages: *pre-authorization*, *proof*, and *verification*. Subsection 3.3.1 covers the pre-authorization stage, that issues a token used to trigger the next stage — this authenticated token holds information like the beacons selected to engage in the proof of location. Subsection 3.3.2 covers the proof stage, where the beacon orchestration takes place — at this point in the process, the prover device is collecting the signals being broadcast by beacons. To conclude, Subsection 3.3.3 covers the verification stage, that culminates with either the acceptance, or the rejection of the location proof generated by the prover.

3.3.1 Pre-authorization stage

A proof of location requires the orchestration of a subset of beacons scattered across a smart location, like an orchestrated room. Beacon orchestration requires some context information to be readily available at proof-time (like which beacons are engaging in the proof of location).

The objective of this stage, represented in Figure 3.7, is to produce an *authorization*, requested by the prover and issued by an orchestrator, that is both:

1. An authenticated and single-use token used to trigger the proof stage;

2. A context information repository that stores relevant metadata necessary for the proof stage.

First, the prover device determines its compatibility with the existing beacon-witness mapping (*Device compatibility check* step). A beacon is deemed supported by the prover device if and only if it has, at least, one compatible untrusted witness (usually a sensor, like a light sensor).

Consequently, the set of supported beacons depends on the hardware properties of the prover device.

The prover device estimates its location resorting to external mechanisms (*Device location estimation* step). For instance, GPS can be used if the signal is strong enough. Upon locating the prover device, GPS coordinates are then converted to a geocode. By default, OLC is used, but other representations are supported (refer to Section 2.2). In the absence of a suitable localization system (like in GPS-constrained environments), scanning an on-site QR code with the geocode of the room might suffice to locate the prover within the building. Naturally, other approaches can be considered, provided they all conjoin to one of the supported geocodes. In the end of this step, the prover requests a proof authorization to an orchestrator (*Request authorization* step).

Following the validation of the request, the orchestrator determines which beacons are available at the location reported by the prover (*Adaptation service discovery* step). Depending upon the smart space topology, different orchestrated rooms offer different beacons, discoverable via adaptation services delivered within that room. To determine which beacons are available at the reported location, the orchestrator:

1. Lists all orchestrated rooms (i.e. the same as listing all existing KAs)
2. Determines the closest one to the prover (by comparing the location returned by each localization service to the location reported by the prover)
3. Discovers which beacons are available in that room (by checking which adaptation services are registered to the KA)

Finally, the orchestrator selects beacons eligible for the proof of location (*Eligible beacons selection* step). A beacon is eligible if and only if (1) it is supported by the prover device, and (2) available at the prover location. If a beacon selection policy is in place, eligible beacons might be narrowed down by the orchestrator to a smaller set, depending upon the policy criteria (e.g. the security level of the room, or the prover's history of behavior). In the end of this step, the orchestrator generates an authorization containing all the relevant information, stores it for future use, and sends it to the prover as a token to trigger the proof stage.

3.3.2 Proof stage

This stage, represented in Figure 3.8, starts when the prover submits the proof authorization to the orchestrator (*Submit authorization* step).

Beacons generate signals based on a set of quirky properties, used to feed a deterministic signal generator (refer to Subsection 3.2.6). Ideally, the same beacon should not broadcast the

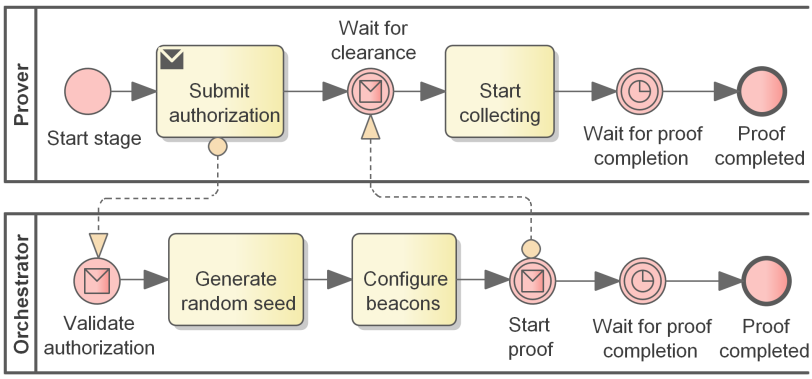


Figure 3.8: Process diagram for the proof stage.

same signal between two consecutive proofs of location. However, that guarantee is too strong, so SureSpace focuses on minimizing the likelihood of that happening by preventing the reuse of the same set of quirky properties. To that extent, a seed value is generated in an unpredictable way (*Generate random seed* step), and used to populate these properties with pseudorandom values derived from it. Naturally, these values are assigned based on the configurable properties declared by the adaptation service of each beacon.

The orchestrator attempts to lock the adaptation services of the selected beacons, and one failed attempt is enough to abort the proof of location (refer to Subsection 3.2.5). If all locks are acquired, the orchestrator applies new pseudorandom values to the quirky properties (*Configure beacons* step). When beacons are ready, they start broadcasting their signal in the vicinities. At the same time, trusted witnesses (virtual or not) in the infrastructure, and untrusted witnesses in the prover device start receiving the signals too.

Recall that witnesses have limited storage capacity, and that the collected information must be stored in time in a different support (refer to Subsection 3.2.6). Untrusted witnesses send the information directly to the prover device, that safely stores the information. On the other hand, trusted witnesses share the information with the orchestrator via the VSL (by updating the corresponding properties of their context models).

Network latency has direct impact on the level of synchronization between orchestrator and prover, since only upon clearance from the orchestrator will the prover initiate the process. If desynchronized, "dead times" may occur in the beginning (beacons are broadcasting, but the prover is waiting for clearance), and in the end of the process (beacons have ceased their activity, but untrusted witnesses remain active). This phenomena *should be* mitigated in the verification stage, without relying on clock synchronization.

3.3.3 Verification stage

Recalling Subsection 3.2.6, the accuracy of the proof ambient is the key factor to determine if a location proof should be accepted or not. Theoretically, a proof of location is accepted if the proof ambient is *complete* enough (with regard to all the signals broadcast by engaging beacons), and *accurate* enough (if the extracted information is in line with the original signal's information).

Necessarily, and as addressed in Subsection 3.2.6, even in optimum conditions, signal degradation will decrease the accuracy of the proof ambient. Internal factors (e.g. witnesses sensitivity, and sensors sampling rate) and external factors (e.g ambient noise, radius of action of the beacon, beacon density, topology of the orchestrated room) might lead to a misrepresented, yet legitimate, proof ambient. To account for such errors, a *margin of error* should be considered when assessing the proof ambient, represented by a tolerance in the comparison of the trusted representation of the original signal with its degraded representation.

To verify the location proof, the prover submits all signal representations to the verifier. A final judgment is yield, either *accepting* or *rejecting* the location proof. To avoid compromising the versatility of the verification step, the implementation details of the stage are left open.

3.4 Summary

In this Chapter, we presented the design rationale of SureSpace — combining the SureThing proof system with the DS2OS dynamic discovery and provisioning of equipments — followed by an in-depth presentation of its architecture, and location certification process. In the next Chapter, we discuss the implementation of the SureSpace prototype.

Chapter 4

SureSpace implementation

This Chapter covers the most relevant details about the implementation of the SureSpace prototype. Section 4.1 explores the development environment of SureSpace, with special emphasis on the tools that were used to implement the different components. Section 4.2 motivates for the need of a formally described technique used by beacons to broadcast their signals, and gives two relevant examples of its application. Section 4.3 identifies beacons currently supported by SureSpace, and goes over the process of adding their full-fledged support. Section 4.4 details how to estimate similarity between two representations of the same signal. Section 4.5 is a brief summary of the Chapter.

4.1 Development platform

The prototype of SureSpace was developed as a Java project, using the JDK (Java Development Kit) version 11. We used Maven 3.6.3 as the build automation tool, that manages the building process, but also does version management for the used libraries. To facilitate dependencies management, the project was organized into different modules.

Module `arduino` implements a custom version of Ardulink 2, a complete, open-source Java solution for the control and coordination of Arduino boards [ard] (this module is required since we used Arduino equipment, as covered in Section 5.1). Module `common` implements everything that needed to be shared between the different modules (like the message structure introduced in Subsection 4.1.1, or the supported location representations listed in Section 2.2). Respectively, modules `ca`, `orchestrator`, and `verifier` implement the functionality of the certificate authority, the orchestrator, and the verifier. Finally, module `ds2os` implements all the DS2OS services (adaptation, localization, and orchestration services) by extending available templates. Some modules were partially inspired in existing work by João Fer-

reira [FP18], but all the code was written from scratch. We used multiple dependencies to implement the project, from cryptographic dependencies (like `org.bouncycastle:bcprov-jdk15on`) or certificate validation dependencies (like `no.difi.commons:commons-certvalidator`), to utility dependencies (like `org.apache.commons:commons-lang3`) or communication dependencies (like `io.grpc:grpc-protobuf` and `io.grpc:grpc-stub`, required by gRPC — refer to Subsection 4.1.1).

To bootstrap the distributed knowledge network, i.e. the VSL, KAs need to be launched one by one, until they become peers of the same network (refer to Section 2.8). In practical terms, a KA is bundled as an executable JAR file, already configured to search for, and join other KAs when launched. However, the default KA was no longer compatible with our setup, so we modified its source code (we were granted access by the DS2OS team), and recompiled it.

The prover was implemented as an Android mobile application, for a richer user-experience. The application was compiled in Java 8, against API 30.

Subsection 4.1.2 identifies the algorithms used for certificate generation and signing, and details identity assignment within the SureSpace domain. Subsection 4.1.1 analyses the underlying protocols used for cross-entity communication, and argues for the need of a common message structure shared by the data flows.

4.1.1 Cross-entity communication

Regarding communication, data transfer flows between entities are bi-directional, and are implemented over different underlying communication protocols. DS2OS entities (both KAs and services) communicate with the VSL via REST connectors, using HTTPS as tunneling protocol for secure communications. All remaining data transfer flows are supported by gRPC (in Java), following a remote invocation paradigm. gRPC uses Protocol Buffers (*protobuf*) to provide a platform-independent representation of remote interfaces, allowing for future clients in different programming languages. It was chosen because of its efficiency, and loose coupling between clients and servers [grp].

In SureSpace, messages share a common payload format, illustrated in Figure 4.1, that (1) allows for a standardized process of message validation, and (2) fosters the implementation of the desired security properties (integrity, authentication, and non-repudiation). To prevent a malicious user from forwarding messages to an entity of their choice, all messages hold information about their source (field `sender`) and destination (field `receiver`) — entity identifiers are used in both fields. Replay attacks are mitigated by the use of a securely generated random number (field `nonce`), unique for each data flow. For integrity, authentication, and non-repudiation purposes, a

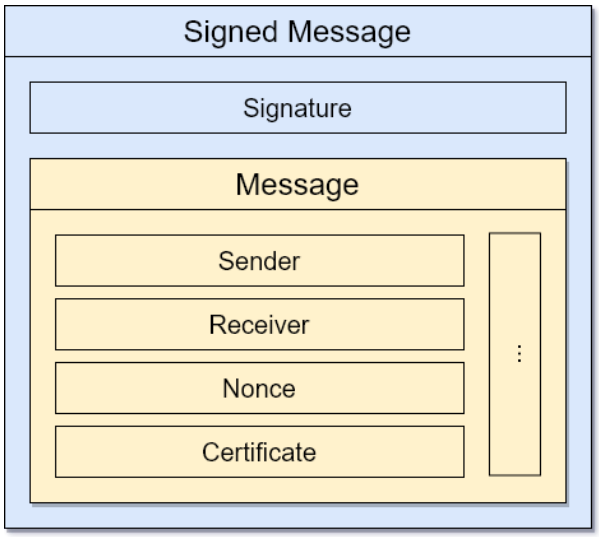


Figure 4.1: Message structure shared in SureSpace.

digital signature is generated over the message (field `signature`) using `SHA-512` with the source entity's private key. To reduce the number of interactions with CAs, the source entity's certificate (field `certificate`), used to validate the signature, is attached to the body of the message prior to signing (the certificate is verified during message validation).

4.1.2 Entity identification

Each SureSpace entity is assigned a public key certificate that is part of a certificate chain that starts with the SureSpace Root CA (refer to Subsection 3.2.2). Public/private key pairs are generated using `2048-bit RSA`, and certificates are signed using `SHA-512`. Moreover, each entity is identified by a hierarchical identifier, unique within the SureSpace domain. That identifier can be directly obtained from the plain text representation of its certificate chain, by (1) mapping certificates in the path into their aliases, and (2) concatenating them using a delimiter, preserving order. For example, `prover1` will be identified by `surespace://rca/pca/prover1` because its certification chain includes the SureSpace Root CA (alias `rca`), the SureSpace Prover CA (alias `pca`), and `prover1`, respectively.

4.2 Beaconing technique

Signals are characterized by a variety of features, and can be classified according to different metrics (their time length, for example). Signal classification allows comparing signals based on a set of common metrics, but not all metrics work for all contexts.

To determine whether a signal is appropriate for a proof of location, we classify it based on two metrics: *difficulty of replication*, and *difficulty of acknowledgment*. Signals should be

difficult to replicate without knowing the quirky properties used for their generation. If signals have noticeable patterns or are reused, a malicious party can easily replicate them, perhaps by trial-and-error. At the same time, signals must be versatile enough to account for common limitations shared by compatible witnesses, so that signals can be easily acknowledged by most witnesses. We introduce a simple technique based on *time fragmentation* to reach an equilibrium between these two metrics.

Subsection 4.2.1 provides a description of the technique, summarized in Algorithm 1. Subsections 4.2.2 describes the application of the technique to a hypothetical light signal. Likewise, Subsection 4.2.3 describes the application of the technique to a hypothetical audio signal.

4.2.1 Technique description

Each signal is broken into a fixed number of consecutive, same-length fragments. Each fragment is generated based on a set of quirky properties, populated with pseudorandom values obtained from a seed. The sequence of fragments constitutes the signal.

For simplicity, we adopt the following notation:

- $b \in B$ denotes beacon b, in the set of supported beacons, B
- $w \in W_b, b \in B$ denotes witness w, in the set of witnesses compatible with beacon b, W_b
- $f_{b,i} \in F_b, b \in B, i \geq 1$ denotes the i-th fragment of the set of fragments that compose a signal broadcast by beacon b, F_b
- $q \in Q_{f_{b,i}}, b \in B, i \geq 1$ denotes quirky property q, in the set of quirky properties used to generate fragment $f_{b,i}$, $Q_{f_{b,i}}$
- $S_b = f_{b,1} \parallel f_{b,2} \parallel \ldots \parallel f_{b,n}, b \in B, i \geq 1$ denotes a signal with $n \geq 1$ fragments, broadcast by beacon b

Algorithm 1 proposes a pseudocode of the technique. This algorithm is executed by the orchestrator in the proof stage (refer to Subsection 3.3.2). In the beginning, a random *seed* is generated in a secure and unpredictable way via function UnpredictableSeed (line 1). Upon that, the orchestrator attempts to lock the adaptation services of all beacons selected for the proof of location (line 3). If the attempt fails, i.e. if, at least, one service fails to be locked, the orchestrator resets the state and aborts the proof of location via function Abort (line 5). On the other hand, if the attempt succeeds, a random number generator is created using the seed (line 9) for each beacon selected for the location proof (line 8). At last, each adaptation service generates their quirky properties via function GenerateQuirkyProperties (line 11). The number

of fragments, $|F_b|$, and the random number generator, *random*, are passed as arguments to that function, that *must* assign a pseudorandom value to each quirky property of each fragment of the signal. If the process fails, the orchestrator aborts via function Abort, already covered (line 13).

Algorithm 1 Time fragmentation technique.

```
1:  seed ← UnpredictableSeed();
2:  try
3:      LockAdaptationServices(B);
4:  catch LockException
5:      Abort();
6:  end try
7:
8:  for all b ∈ B do
9:      random ← Random(seed);
10:     try
11:         GenerateQuirkyProperties(|F_b|, random);
12:     catch GenerationException
13:         Abort();
14:     end try
15: end for
```

In the next Subsections, we apply the technique to two representative signals: a light signal, and an audio signal. Moreover, for each case, we provide the pseudocode for function GenerateQuirkyProperties, executed by the respective adaptation services.

4.2.2 Light signal time fragmentation

Consider a light source (e.g. a LED) used as a beacon in a proof of location, b_{light}, that can switch between states `on` and `off` with period $P \in [P_{MIN}, P_{MAX}[$, measured in a convenient unit (P_{MIN} and P_{MAX} are, respectively, the lowest and the highest supported periods). In the proof stage, the beacon broadcasts a signal, S_{light}, split into n d-seconds fragments. During each fragment $f_{light,i},\ i = 1, 2, \ldots, n$, the beacon switches between states with a pseudorandom period P_i (the quirky property), derived from the seed, forming a *power-on and power-off* sequence with rates that vary between fragments.

Theoretically, P_i can take any value in range $[P_{MIN}, P_{MAX}[$, and nothing prevents two pseudorandom periods from being equal or close enough to generate similar or indistinguishable fragments. To avoid this, we propose an approach that (1) prevents reusing the same period and (2) reduces the probability of picking periods too close in the range.

At a higher level, the initial range is split into n subranges of equal length, shuffled using the seed, and exactly one period is pseudorandomly picked from each subrange, in a total of n

periods (one per fragment). Algorithm 2 provides a detailed explanation of our approach. The adaptation service creates an empty array to store the n pseudorandom periods (line 2) — array indices start at 1 for simplicity. The subranges length, *length*, is computed by dividing the length of the initial range, given by $P_{MAX} - P_{MIN}$, by the number of fragments, $|F_{b_{light}}|$ (line 3). Upon that, a unique subrange is computed (lines $6-7$) for each fragment (line 5), and the union of all subranges equals the initial range, according to Equation 4.1.

$$\begin{aligned}[P_{MIN}, P_{MAX}[= &[P_{MIN}, P_{MIN} + length[\\ &\cup [P_{MIN} + length, P_{MIN} + 2 \times length[\cup \\ &\dots \\ &\cup [P_{MIN} + (n-1) \times length, P_{MAX}[\end{aligned} \tag{4.1}$$

Then, a pseudorandom period is picked between the lower bound of the subrange (inclusive) and its upper bound (exclusive), drawn from the random number generator's sequence (line 9). Each pseudorandom period is added to the array in the right position (line 10), and the array is shuffled in the end, using the random number generator (line 13). If the array was not shuffled, periods would increase sequentally between fragments (i.e. $P_{i+1} > P_i$ would hold true for all periods), creating an undesired pattern in the light signal. Finally, each quirky property $P_i, i = 1, 2, \dots, n$ is assigned the i-th pseudorandom period (line 15).

Algorithm 2 Generate quirky properties for a light signal.

```
1:  procedure GenerateQuirkyProperties(|F_{b_light}|, random)
2:      periods = [];
3:      length = (P_MAX − P_MIN) / |F_{b_light}|;
4:
5:      for i ← 1, 2, ..., |F_{b_light}| do
6:          lower = P_MIN + (i − 1) × length;
7:          upper = lower + length;
8:
9:          period = random.next(upper − lower) + lower;
10:         periods[i] = period;
11:     end for
12:
13:     periods.shuffle(random);
14:     for i ← 1, 2, ..., |F_{b_light}| do
15:         P_i ← periods[i];
16:     end for
17: end procedure
```

Regarding its nature, S_{light} is visible light, produced by the light source from electric current. Light can be described as either a particle or a wave, and has different properties that can be

acknowledged depending on the receiver. A light sensor, for instance, can measure the light intensity, but cannot determine what color the light is. A spectroradiometer, on the other hand, is able to measure the wavelength of the light (which can, then, be interpreted as a color).

For this purpose, consider a light sensor used as a witness in the same proof of location, w_{light}, capable of measuring light intensity in a convenient unit, at a sampling rate not less than P_i^{-1}, $\forall i$. Plotting the measurements over time offers a representation of S_{light} based on one of its properties (light intensity). The analysis of that representation may confirm significant variations in light intensity, which are an interpretation of the power-on and power-off sequence.

It is important to mention that P_{MIN} and P_{MAX} should be picked carefully. These values *must* be adequate considering (1) the length of the light signal and (2) the sampling rates of witnesses expected to receive the signal. If P_{MIN} is too low, witnesses may miss variations in light intensity that occur in-between samples. Conversely, if P_{MAX} is too high, witnesses may not detect any variation in light intensity during the fragment.

4.2.3 Audio signal time fragmentation

Consider an audio source (e.g a speaker) used as a beacon in a proof of location, b_{audio}. The beacon can be programmed to play any song out of a predefined set of m songs, and $songId \in \{0, 1, \ldots, m-1\}$ is the index of the song to be played. This song is any regular song that plays on the radio, and we consider it over any synthesized melody because a song is more easily tolerated by the human ear for extended periods of time. During its activity, the beacon broadcasts a signal, S_{audio}, with a single d-seconds fragment, and a pseudorandom $songId \in Q_{f_{b_{audio}},1}$ (the only quirky property) is derived from the seed.

According to Algorithm 3, the adaptation service generates a pseudorandom song index, between 0 (inclusive) and m (exclusive), drawn from the random number generator's sequence and assigned to $songId$ (line 2).

Algorithm 3 Generate quirky properties for an audio signal.

```
1: procedure GenerateQuirkyProperties(|F_{b_audio}|, random)
2:     songId = random.next(m);
3: end procedure
```

Regarding its nature, S_{audio} is a sound, produced by the audio source that converts an electric audio signal into a corresponding sound. Sound is propagated as an acoustic wave, often simplified to sinusoidal waves characterized by amplitude, frequency, speed, and direction. Not all properties can be determined with the same ease. For instance, finding the direction of an acoustic wave requires acoustic source localization, usually applying a TDoA technique (refer to

Subsection 2.4.4) between couples of acoustic sensors [MNV03]. On the other hand, amplitude can be measured using a sound sensor.

For this purpose, consider a sound sensor used as a witness in the same proof of location, w_{audio}, capable of measuring sound amplitude in a convenient unit. Plotting the measurements over time offers a representation of S_{audio} based on one of its properties (audio amplitude). The analysis of that representation may confirm variations in amplitude and frequency that match the song being played.

4.3 Supported beacons

Adding support for a new beacon in SureSpace requires writing its context model with all the configurable properties (the trusted witness requires its own context model too), and implementing the adaptation service, so that the beacon can be discoverable. Furthermore, to be eligible for the proof of location, the beacon must have, at least, one corresponding untrusted witness in the prover device (refer to Subsection 3.2.6). Thus, although implicit, adding support to a beacon also requires having, at least, one compatible witness in the prover device, so that the constructed representations can be correctly compared.

This Section goes over these steps to add support for a light beacon (refer to Subsection 4.3.1), and an audio beacon (refer to Section 4.3.2).

4.3.1 Light beacon and witnesses

Based on the example in Subsection 4.2.2, we considered a light beacon capable of switching between states `on` and `off` with a configurable period. Figure 4.2 is a simplified context model of the beacon, where `isOn` is a boolean used to control the beacon (if set to `true`, the beacon is working), and `switchingPeriod` is the time it takes for the beacon to switch between states (in seconds).

```
1 <model type="/complex/beacon">
2     <isOn type="/basic/boolean"/>
3     <switchingPeriod type="/basic/number"/>
4 </model>
```

Figure 4.2: Simplified context model of a light beacon.

As trusted witness, we considered a light sensor capable of measuring light intensity at a configurable sampling rate not less than $\texttt{switchingPeriod}^{-1}$. Figure 4.3 is a simplified context

model of the trusted witness, where `intensity` is the measured light intensity (in a convenient unit), `intensitySamplingRate` is the sampling rate at which the sensor is reading (in Hertz), and `isOn` is a boolean used to control the witness (if set to `true`, the witness is working). The orchestration service subscribes the `intensity` attribute on the trusted witness context model. Every time the attribute value changes because of a new measurement, the orchestration service is notified. At that moment, the value is timestamped (in milliseconds), and stored in the orchestrator to compose the trusted representation of the light signal.

```
<model type="/complex/witness">
    <intensity type="/basic/number"/>
    <intensitySamplingRate type="/basic/number"/>
    <isOn type="/basic/boolean"/>
</model>
```

Figure 4.3: Simplified context model of a light witness.

As untrusted witness, we consider any device capable of measuring light intensity to ensure both witnesses acknowledge the same property of the signal (intensity).

4.3.2 Audio beacon and witnesses

Based on the example in Subsection 4.2.3, we considered an audio beacon capable of playing a specific song out of a set of songs stored in a raw format (like WAV). Figure 4.4 is a simplified context model of the beacon, where `isOn` is a boolean used to control the beacon (if set to `true`, the beacon is working), and `songId` is the index of the song to be played.

```
<model type="/complex/beacon">
    <isOn type="/basic/boolean"/>
    <songId type="/basic/number"/>
</model>
```

Figure 4.4: Simplified context model of an audio beacon.

As trusted witness, we considered a sound sensor capable of measuring the sound amplitude at a specified sampling rate. Figure 4.5 is a simplified context model of the trusted witness, where `amplitude` is the measured sound amplitude (in a convenient unit), `amplitudeSamplingRate` is the sampling rate at which the sensor is reading (in Hertz), and `isOn` is a boolean used to control the witness (if set to `true`, the witness is working).

The trusted witness was implemented as fully virtual to demonstrate the feasibility of the approach. Since we have access to the songs in a raw format, it is possible to emulate the behavior of a physical witness. Every `amplitudeSamplingRate`$^{-1}$ s, the virtual witness reads the sound amplitude from the file, and updates the `amplitude` attribute on its context model, which has been subscribed by the orchestration service. Every time the attribute value changes, the orchestration service is notified. At that moment, the value is timestamped (in milliseconds), and stored in the orchestrator to compose the trusted representation of the audio signal.

```
<model type="/complex/witness">
    <amplitude type="/basic/number"/>
    <amplitudeSamplingRate type="/basic/number"/>
    <isOn type="/basic/boolean"/>
</model>
```

Figure 4.5: Simplified context model of an audio witness.

As untrusted witness, we consider any device capable of measuring sound amplitude to ensure both witnesses acknowledge the same property of the signal (amplitude).

4.4 Verifier implementation

The verifier measures the accuracy of the proof ambient to determine the legitimacy of a location proof (refer to Subsection 3.2.6). In other words, it quantifies similarity between different representations of the same signal: a degraded one (from untrusted witnesses in the prover device), and a trusted one (from trusted witnesses in the infrastructure). If more than one beacon is used (and, thus, more than one signal is involved), these values need to be weighted, since different signals may contribute differently to the overall accuracy of the location proof.

In our context, based on the set of supported beacons, we use the MATLAB Engine API for Java to quantify similarity. For signal processing, we require additional toolboxes, namely the Signal Processing Toolbox, the Statistics and Machine Learning Toolbox, and the Communications Toolbox [mat].

This Section details the approaches used for comparing representations of the same signal, for both light signals (refer to Subsection 4.4.1), and audio signals (refer to Subsection 4.4.2). Moreover, it details how multiple signals are handled by the verifier (refer to Subsection 4.4.3).

4.4.1 Light signal representations similarity estimation

Representations may be sampled at different rates, impeding their comparison. To bring them to a common rate, we upsample the representation with the lowest frequency, using linear interpolation. This process produces an approximation of the representation that would have been obtained by sampling at a higher rate. Upsampling does not change the accuracy of the representation itself, since new samples are estimated based on existing ones.

Since clock synchronization between the prover and the system is not a requirement, a potential delay between representations may exist. To align them without relying on timestamps, we use correlation to determine where representations overlap the most, and then align them.

At last, we normalize both representations, and calculate the linear correlation coefficient between them, $corr_{light\ 1}$, given by Equation 4.2

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4.2}$$

where r is the linear correlation coefficient, n is the number of samples in the representations (they have the same size), x_i and y_i are the sample points, and $\bar{x}$ and $\bar{y}$ are the means of the samples. This coefficient measures the linear relationship between the two representations, and is used as an estimate of their similarity.

4.4.2 Audio signal representations similarity estimation

Just like in the case of light representations, audio representations may be sampled at different rates, and may not be aligned, impeding their direct comparison. However, audio and light signals are fundamentally different, and so are their representations. For that reason, we cannot follow the previous approach. Instead, after normalization, we use dynamic time warping to resample and align the representations. This algorithm stretches the two representations onto a common set of instants, such that the sum of the Euclidean distances between corresponding points is smallest. Then, we calculate the linear correlation coefficient between the aligned representations, $corr_{audio\ 1}$, and use it as a first estimate of their similarity.

To improve the estimate, we calculate the power spectrum of the two representations. Simply put, a power spectrum is a frequency-domain interpretation of an audio signal representation because it describes the distribution of power (sound amplitude) into frequency components. In this context, this information is relevant because each song has a different time-frequency structure. Thus, we calculate the linear correlation coefficient between the power spectra of the two representations, $corr_{audio\ 2}$, and use it as a second estimate of their similarity.

The two estimates are weighted for a final similarity estimate, given by Formula 4.3

$$w_1 \times corr_{audio\ 1} + w_2 \times corr_{audio\ 2}, w_2 = 1 - w_1 \tag{4.3}$$

Weights w_1 and w_2 must be tuned based on a training set, since they are necessarily beacon- and witness-dependent.

4.4.3 Combined signals similarity estimation

In the presence of more than one signal, individual similarity estimates are weighted for a final similarity estimate, given by Formula 4.4

$$w_3 \times corr_{light\ 1} + w_4 \times (w_1 \times corr_{audio\ 1} + w_2 \times corr_{audio\ 2}), w_4 = 1 - w_3 \tag{4.4}$$

This value is the final estimate of the accuracy of the proof ambient, and, thus, it is the value used to determine if a location proof should be accepted. Once again, weights w_3 and w_4 need to be tuned, as discussed in Subsection 4.4.2.

4.5 Summary

In this Chapter, we started by describing the development environment of SureSpace, going over the tools that were used to implement the different components. We detailed how entities are identified in SureSpace, and how they communicate through different underlying protocols. We presented a simple technique based on time fragmentation that allows beacons to broadcast complex signals, and gave two representative examples of the application of the technique. We explained how beacons become supported by SureSpace, and provided details about the implementation of two beacons. Finally, we proposed a simple approach to estimate similarity between two representations of the same signal. The developed prototype is evaluated in the next Chapter.

Chapter 5

Evaluation

In this Chapter, we present the experimental setup used to evaluate the SureSpace prototype, describe the evaluation criteria, and discuss the evaluation results. Section 5.1 describes the orchestrated area, the equipment used for the proofs of location (both in the infrastructure, and in the prover device), and explains the evaluation criteria used to assess the prototype. Section 5.2 details how important parameters were tuned to maximize the accuracy of the prototype, and justifies the approach used to conduct the experiments. Section 5.3 addresses the effectiveness of SureSpace, discussing relevant examples of some location proofs. Finally, Section 5.4 measures the performance of the prototype under attack.

5.1 Experimental setup

Conceptually, our approach fits any orchestrated area, regardless of its physical traits (e.g. blueprint complexity, or coverage area), provided that the equipment being used for beacons and witnesses is adequate. For the experimental setup, we used inexpensive equipment, with less accuracy, but more representative of commodity equipment that we expect to find in a smart building. We dropped room-level orchestration, and opted for a smaller, yet representative, orchestrated area shown in Figure 5.1.

Recall the notation introduced in Subsection 4.2.1, where b denotes a supported beacon, and w a corresponding witness. Based on the set of supported beacons (refer to Section 4.3), we used a Grove Chainable RGB Led V2.0 as light beacon, b_{light}, a Grove Light Sensor V1.2 as trusted light witness, w_{light}, and a JBL GO 2 Speaker, connected to a Grove MP3 V2.0 module, as audio beacon, b_{audio}. The trusted audio witness, w_{audio}, is fully virtual, eliminating the need for physical equipment (refer to Subsection 4.3.2). For connectivity reasons, b_{light}, w_{light}, and the Grove MP3 V2.0 module were all connected to a Grove Base Shield V2.0 for an Arduino

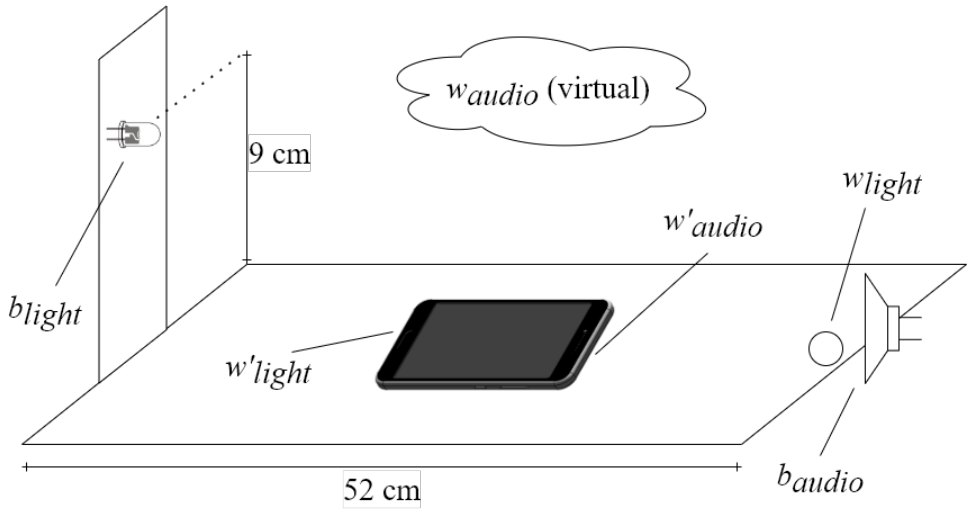


Figure 5.1: Experimental setup components and orchestrated area.

Uno board. The prover device was a Huawei Mate 20 Pro Android smartphone, shipped with Android 10, equipped with a built-in ambient light sensor, the untrusted light witness, w'_{light}, and a microphone, the untrusted audio witness, w'_{audio}. During the proof of location, the prover device is steady in the center of the orchestrated area, as depicted.

We built a dataset by running 80 location proofs under the same representative *controlled scenario*. Each location proof delivered *four* signal representations (two representations per signal, a trusted one and an untrusted one). To be assessed, a location proof is submitted to the verifier to estimate the similarity between the two light signal representations (refer to Subsection 4.4.1), and the similarity between the two audio signal representations (refer to Subsection 4.4.2). These two estimates are weighted for a final similarity estimate, that leads either to the acceptance or to the rejection of the location proof (refer to Subsection 4.4.3).

In some cases, the verifier may classify location proofs incorrectly, either by accepting a location proof that should be rejected (false positive), or, conversely, by rejecting a location proof that should be accepted (false negative). The accuracy of this judgment, based on the number of incorrect classifications, is a good metric to evaluate SureSpace. In particular, we consider the false positive rate (FPR), which is the percentage of all negatives that still yield positive, the false negative rate (FNR), which is the percentage of positives that yield negative, and the success rate, which is the percentage of correct classifications. Intuitively, FPR and FNR are inversely proportional to the success rate. Thus, ideally, the verifier should focus on minimizing both FPR and FNR.

To ensure the reproducibility of the experiments, the duration of the proof stage was set to 30 seconds in all proofs of location (a reasonable value in a human time-scale that works in the meeting room scenario we presented in the beginning of Chapter 3). Regarding the

audio component, beacon b_{audio} could choose between 20 different predefined songs, all sampled at 44.1 kHz (refer to Subsection 4.2.3). Regarding the light component, light signals were broken into two 15-seconds fragments, and beacon b_{light} could switch between states with a pseudorandom period $P \in [0.5, 7.5[$ (refer to Subsection 4.2.2). The upper bound of the range is $7.5s$ $(= \frac{15s}{2})$ to ensure that light signal fragments generate, at least, one complete `on-off` sequence (i.e. the beacon is powered on, and then it is powered off for the same amount of time at least once). The lower bound of the range is $0.5s$ to ensure compatibility with all light witnesses (based on the information we collected, w'_{light}, the Android ambient light sensor, is the light witness that reports the lowest sampling rate of $\approx 2Hz$).

5.2 Optimal weight tuning

As addressed in Section 4.4, weight tuning is required for our experimental evaluation. We started by dividing our dataset into two subsets: a training set, and a test set. Following the Pareto principle, the training set accounted for 20% of the dataset (i.e. 16 location proofs), and the test set accounted for the remaining (i.e. 64 location proofs). This way, we ensured the test set was large enough to yield statistically meaningful results, and was representative of the dataset as a whole (because all location proofs shared the same configuration parameters).

In the first place, and to be able to estimate similarity between two audio signal representations, we need to tune weights w_1 and w_2 using the training set (refer to Subsection 4.4.2). We crossed audio signal representations from all location proofs in the training set, ending with 256 combinations (16×16), from which 16 should be accepted (the number of legitimate location proofs), and 240 should be rejected (the number of fabricated location proofs). In small steps of 0.001, we varied w_1 (recall that $w_2 = 1 - w_1$) to find the optimal combination that would minimize $FPR + FNR$. Figure 5.2 plots the sum as a function of w_1. The local minima is at $w_1 = 0.661$, which means that $\langle w_1, w_2 \rangle = \langle 0.661, 0.339 \rangle$ offers the best success rate ($FPR = 33.33\%$ and $FNR = 37.50\%$). Replacing w_1 and w_2 in Formula 4.3, the audio signal representations similarity estimate is given by Formula 5.1

$$0.661 \times corr_{audio\ 1} + 0.339 \times corr_{audio\ 2} \tag{5.1}$$

Then, we tuned weights w_3 and w_4 to be able to estimate the final similarity (refer to Subsection 4.4.3), that would determine if a location proof is either accepted or rejected. Following the same approach, we crossed audio and light signals from all location proofs in the training set, and varied w_3 (recall that $w_4 = 1 - w_3$) to find the optimal combination that would minimize

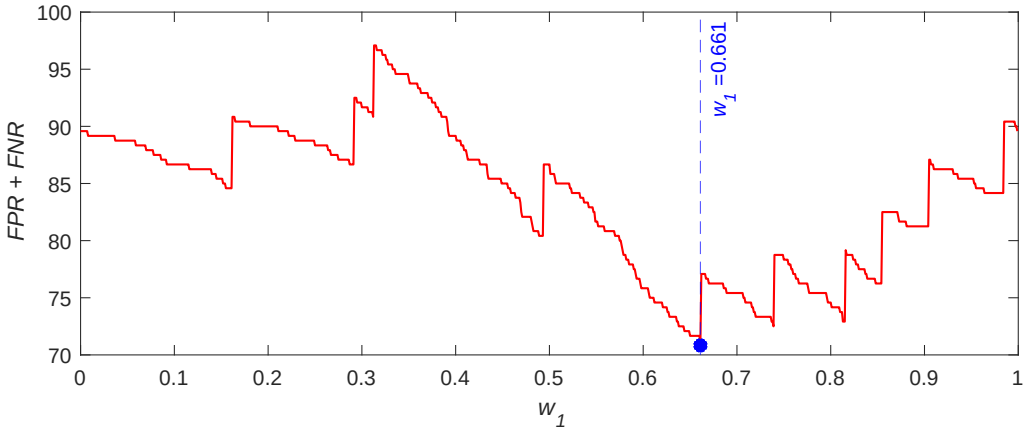


Figure 5.2: Optimal value of w_1.

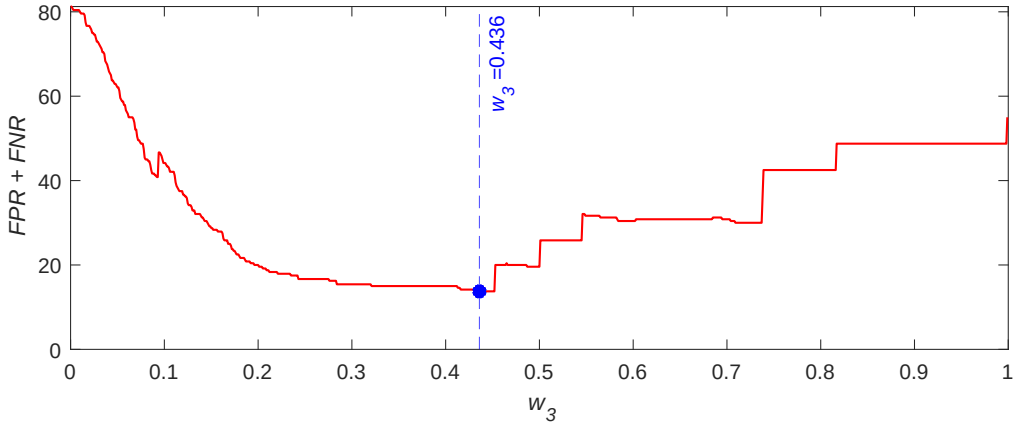


Figure 5.3: Optimal value of w_3.

$FPR + FNR$. Figure 5.3 plots the sum as a function of w_3. The local minima is at $w_3 = 0.436$, which means that $\langle w_3, w_4 \rangle = \langle 0.436, 0.564 \rangle$ offers the best success rate ($FPR = 7.50\%$ and $FNR = 6.25\%$). Replacing all weights in Formula 4.4, the final similarity estimate is given by Formula 5.2

$$0.436 \times corr_{light\ 1} + 0.564 \times (0.661 \times corr_{audio\ 1} + 0.339 \times corr_{audio\ 2}) \tag{5.2}$$

5.3 Approach effectiveness

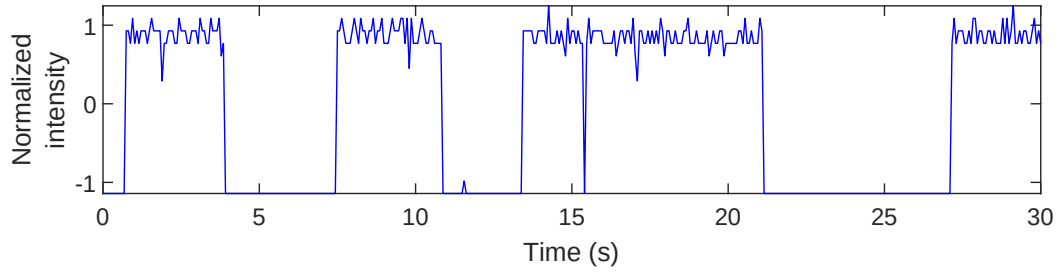
Next, we validated our approach by testing our signal representation similarity estimate against the test set. We crossed light and audio signal representations from all location proofs in the test set, ending with 4096 combinations (64×64), from which 64 should be accepted (the number of legitimate location proofs), and 4032 should be rejected (the number of fabricated location proofs). In the end, the verifier classified location proofs correctly in 94.78% of the cases, with

rates $FPR = 5.06\%$ and $FNR = 15.63\%$.

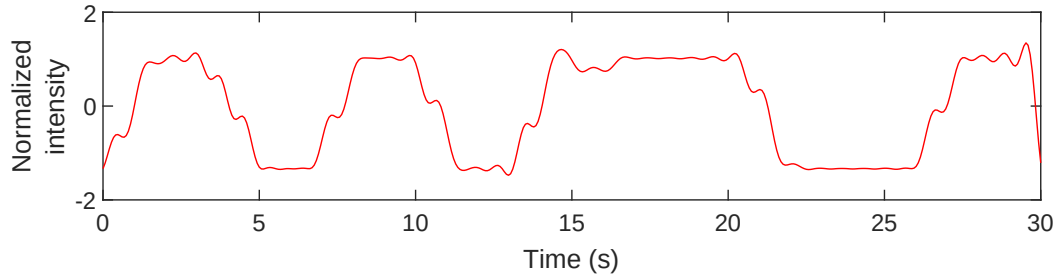
For demonstration purposes, consider the two light signal representations of a location proof accepted by the verifier. Figures 5.4a and 5.4b plot the normalized light intensity read by the trusted witness, and the untrusted witness, respectively, upon the processing steps described in Subsection 4.4.1. It becomes evident that the light signal is split into two fragments, with the second fragment starting at around 15 s. At that moment, the rate at which b_{light} changes between states `on` and `off` decreases. However, Figure 5.4a presents sharp variations in light intensity, with some noticeable spikes, while Figure 5.4b presents smoother variations. This difference can be explained by the very different sampling rates at which both witnesses work. Based on the information we collected, w_{light} (the Arduino light sensor) works at $\approx 14\ Hz$, while w'_{light} (the Android ambient light sensor) works at $\approx 2\ Hz$. For that reason, the trusted witness is aware of the commence of the second fragment, while the untrusted witness misses it, since it occurs in-between samples. Notwithstanding, Figure 5.4c emphasizes how both representations overlap, suggesting both witnesses were exposed to the same light conditions and, thus, engaged in the same proof of location.

Additionally, consider the two audio signals representations of a location proof accepted by the verifier. Figures 5.5a and 5.5b plot the normalized sound amplitude read by the virtual trusted witness, and the untrusted witness, respectively, upon the processing steps described in Subsection 4.4.2. As described in Subsection 4.2.3, the signal is not split into multiple fragments, because the song being played already creates a pattern in sound amplitude by itself. Although both representations overlap, as emphasized in Figure 5.5c, there is a "clear" difference between both representations. Figure 5.5a presents stronger and neater variations in sound amplitude, while Figure 5.5b looks like a sketchy version of the trusted representation, with noticeable spikes and periods of constant sound amplitude. Once again, this is a consequence of sampling the same sound at different sampling rates. Based on the information we collected, w_{audio} (the virtual trusted witness) works at $\approx 30\ Hz$, while w'_{audio} (the Android microphone) works at $\approx 12\ Hz$. The power spectra of both representations, depicted in Figure 5.5d, shows prominent peaks in magnitude occurring around the same frequencies. This suggests that both witnesses were capturing the same song being played by b_{audio}, since each song has a different time-frequency structure.

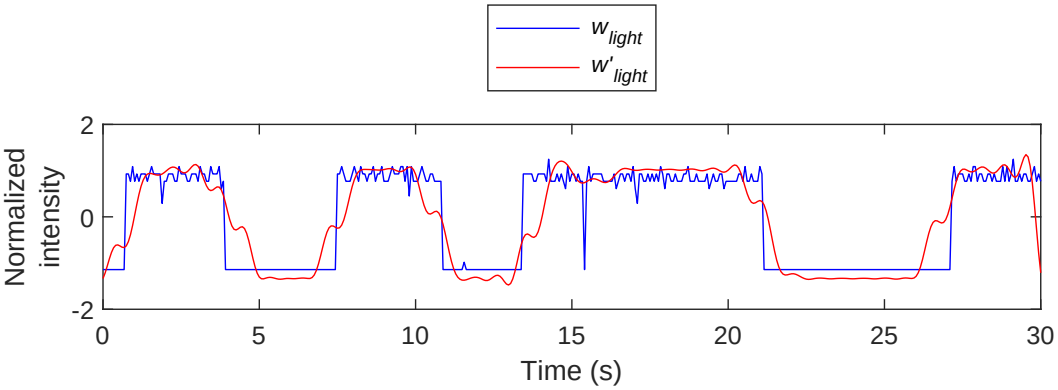
In Section 5.2, we estimated the optimal values for weights w_1 and w_2, and concluded that $w_1 > w_2$. In practical terms, and in the context of the training set, $corr_{audio\ 1}$ (calculated from data in Figure 5.5c) offers a better similarity estimate than $corr_{audio\ 2}$ (calculated from data in Figure 5.5d). Furthermore, we concluded that $w_4 > w_3$, meaning that the audio signal similarity



(a) Light signal representation by w_{light}.

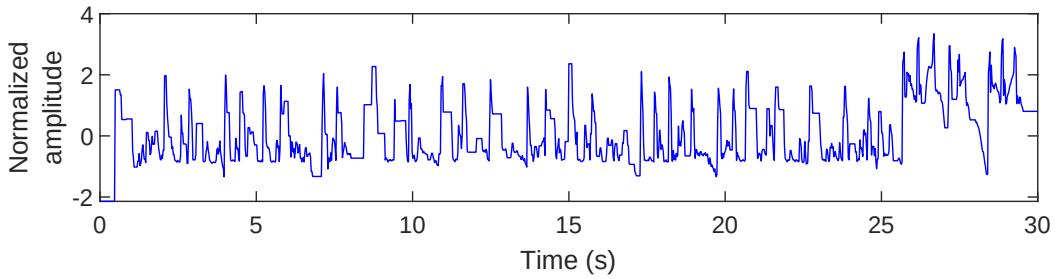


(b) Light signal representation by w'_{light}.

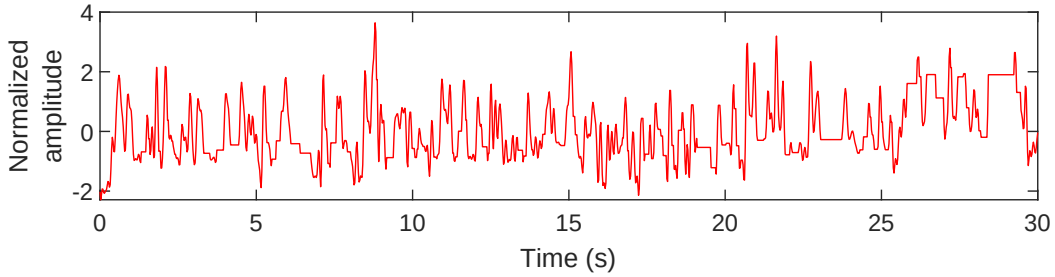


(c) Light signal representations overlapping.

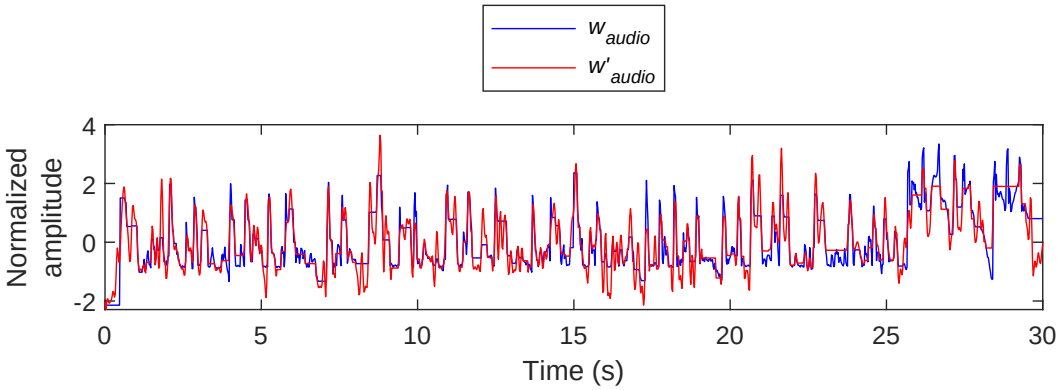
Figure 5.4: Trusted and untrusted light signal representations comparison.



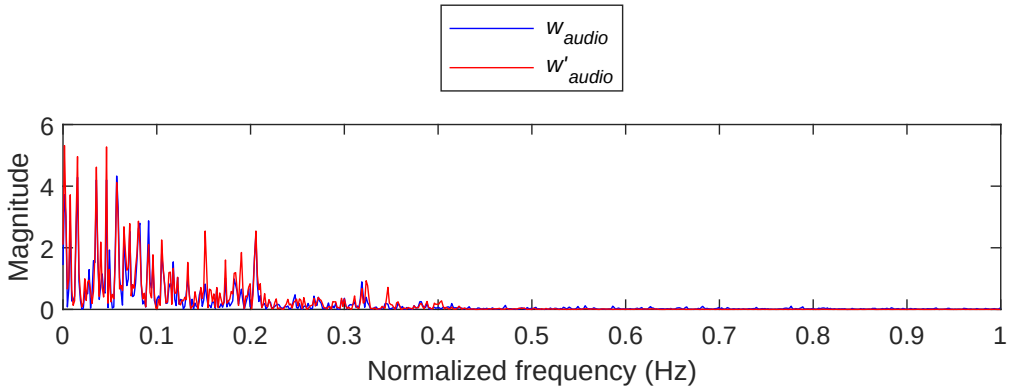
(a) Audio signal representation by w_{audio}.



(b) Audio signal representation by w'_{audio}.



(c) Audio signal representations overlapping.



(d) Audio signal representations spectra overlapping.

Figure 5.5: Trusted and untrusted audio signal representations comparison.

estimate is the decisive factor when accepting or rejecting a location proof. However, because $w_3 \neq 0$, the light signal similarity estimate is a complementary information that helps the verifier by increasing the success rate.

5.4 Resistance to attacks

At a higher level, we can consider that SureSpace has successfully resisted an attack when it rejects an illegitimate location proof. From another angle, we care about the number of false positives, which represent a situation where a location proof fabricated by an attacker is accepted when it should have been rejected. Thus, we consider the FPR as the only metric to evaluate SureSpace's resistance to attacks (as defined in Section 5.1).

Based on the low-level capabilities of a SureSpace attacker (refer to Chapter 3), we consider two attackers that try to prove their false presence by manipulating signals:

A1 Receives the signal from exactly one random beacon, but not from the others (since beacons can have different radius of action);

A2 Combines legitimate signals representations to derive a synthesized proof ambient from them.

Since we used two beacons in our experiments, attacker A1 can either (1) receive the light signal but not the audio signal, or (2) receive the audio signal but not the light signal. Based on Formula 5.2, used to determine if a location proof should be accepted, we confirm that signals have similar weights (0.436 vs 0.564), so we expect location proofs to be rejected if an attacker only provides the representation of one of the signals. To simulate attacker A1, we used the 4032 fabricated location proofs that should be rejected by the verifier (refer to Section 5.3), and, depending on the scenario, (1) or (2), we discarded one of the signal representations when sending the location proof to the verifier. In both scenarios, we obtained $FPR = 0.00$ %, meaning that the verifier successfully rejected all location proofs fabricated by the attacker.

In Section 5.3, we combined legitimate signal representations from different location proofs to fabricate new illegitimate location proofs. In fact, we were already simulating attacker A2, and determined $FPR = 5.06$ %. We believe that the technique introduced in Subsection 4.2.1 contributed to increase SureSpace's resilience against this type of attacks. Because signals are split into different fragments, and each fragment has unique characteristics within the same signal, the likelihood of broadcasting the same signal twice is low. For that reason, combining signals from different location proofs is not an effective attack against SureSpace, because the verifier is capable of telling they were generated for different location proofs.

Chapter 6

Conclusion

Location certification systems have been proposed to counterbalance the lack of trustworthy mechanisms that provide a user's real location. Existing systems, like SureThing, resort to witnesses that attest to the presence of a prover with varying degrees of trustworthiness. Meanwhile, smart spaces have evolved to become an enabling environment for the exchange of information between heterogeneous smart devices. When interconnected, they can be programmed to deliver valuable context-aware services to the final user. However, interoperability issues between these devices pose a problem to the orchestration capabilities of smart spaces. Orchestration systems, like the DS2OS, focus on improving these capabilities by providing a framework to discover and control these smart devices with relative ease, regardless of their specifics.

6.1 Achievements

In this work, we presented SureSpace, a location certification system designed for smart environments. It leverages the capabilities of both SureThing, that defines procedures and techniques for proofs of location, and DS2OS, that provides control over diverse smart devices for orchestration purposes. SureSpace relies on smart devices that broadcast preconfigured signals, and witnesses that capture these signals, to certify location at a specific time and place, without requiring the presence of other users. The system was evaluated in laboratory conditions with inexpensive Arduino-compatible equipment. It was shown to be effective using light and audio signals, with a success rate up to 94.78%. Moreover, we evaluated SureSpace's resistance to attacks by simulating the capabilities of rogue provers, that try to prove their false presence.

6.2 Future work

In its current version, the success rate of SureSpace depends greatly on the weights used to estimate similarity between signal representations. Our primary objective was to show the feasibility of the idea, so we used a basic approach for optimal weight tuning that yielded good results. As future work, we consider the possibility of implementing more complex and robust methods for optimal weight tuning, that could possibly improve the effectiveness of SureSpace.

Also, we consider a new approach to verify location proofs. Currently, to assess a location proof, the prover submits all the collected information to the verifier, that quantifies similarity between different representations of the same signal. Instead, we could follow a *challenge-response* approach to simplify the process while preserving its success rate. In this scenario, the verifier challenges the prover about a specific location proof, by asking a question whose answer can be deduced from the proof ambient. For example, and considering the light signal, we could consider questions like "For how long was the beacon in the `on` state?". Or even "Was the beacon switching between states faster in the first fragment?". This allows the participation in a proof of location without disclosing the complete captured signal.

In SureSpace, provers are assigned identifiers that do not change over time, becoming long-term identifiers. They can be used to distinguish between the creators of different location proofs. If a malicious party gains access to the record of past location proofs, sensitive information (like time, location, and even device specifications — based on the set of beacons supported by the prover device) can be disclosed. To prevent this, we suggest the implementation of an adapted version of the privacy protection mechanism proposed in [BPB13]. Provers are still assigned long-term identifiers by a *long-term CA*. These identifiers are not used in regular communications, and must not be disseminated to the public. Instead, provers use short-term identifiers, known as *pseudonyms*, assigned by a *pseudonym CA*, that change regularly over time, preserving the real identity. In some cases, however, resolving the pseudonym (i.e. retrieving the appropriate long-term identifier from it) should be possible (e.g. when the orchestrator wants to validate the prover identity). To address this, the mechanism proposes a protocol that allows pseudonym resolution under defined and controllable conditions. This mechanism can be added to a future version of SureSpace.

Finally, SureSpace could be evaluated in an actual smart environment, and, later, in a smart building, using available smart devices, like smart bulbs, and smart speakers. The adoption of SureSpace would also require the development of an improved end-user mobile application. Then, it could be used in real-world applications, like a check-in app for physical meetings.

References

[ACD+06] Claudio A. Ardagna, Marco Cremonini, Ernesto Damiani, Sabrina De Capitani di Vimercati, and Pierangela Samarati. Supporting Location-Based Conditions in Access Control Policies. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '06, page 212–222, New York, NY, USA, 2006. Association for Computing Machinery.

[AL19] Georg Aures and Christian Lübben. DDS vs. MQTT vs. VSL for IoT. *Network*, 1, 2019.

[ard] Ardulink 2. `https://github.com/Ardulink/Ardulink-2`. Accessed: 2020-12-21.

[BK11] Rahul C Basole and Jürgen Karla. On the evolution of mobile platform ecosystem structure and strategy. *Business & Information Systems Engineering*, 3(5):313, 2011.

[BMBB14] AH Buckman, Martin Mayfield, and Stephen BM Beck. What is a smart building? *Smart and Sustainable Built Environment*, 3(2):92–109, 2014.

[BPB13] Norbert Bißmeyer, Jonathan Petit, and Kpatcha M Bayarou. CoPRA: Conditional pseudonym resolution algorithm in VANETs. In *2013 10th annual conference on wireless on-demand network systems and services (WONS)*, pages 9–16. IEEE, 2013.

[CCCDP13] Eyüp S Canlar, Mauro Conti, Bruno Crispo, and Roberto Di Pietro. Crepuscolo: A collusion resistant privacy preserving location verification system. In *2013 International Conference on Risks and Security of Internet and Systems (CRiSIS)*, pages 1–9. IEEE, 2013.

[CP20] João Costa and Miguel L. Pardal. A Witness Protection for a Privacy-Preserving Location Proof System, 2020.

[CPTT18] Mario Collotta, Giovanni Pau, Timothy Talty, and Ozan K Tonguz. Bluetooth 5: A concrete step forward toward the iot. *IEEE Communications Magazine*, 56(7):125–131, 2018.

[FP18] João Ferreira and Miguel L. Pardal. Witness-based location proofs for mobile devices. In *17th IEEE International Symposium on Network Computing and Applications (NCA)*, November 2018.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[grp] About gRPC | gRPC. `https://www.grpc.io/about/`. Accessed: 2020-12-23.

[HC16] Qiwei Han and Daegon Cho. Characterizing the technological evolution of smartphones: insights from performance benchmarks. In *Proceedings of the 18th Annual International Conference on Electronic Commerce: e-Commerce in Smart connected World*, pages 1–8, 2016.

[HGK+18] Haosheng Huang, Georg Gartner, Jukka M Krisp, Martin Raubal, and Nico Van de Weghe. Location based services: ongoing evolution and research agenda. *Journal of Location Based Services*, 12(2):63–93, 2018.

[HHFM17] Daniel Hintze, Philipp Hintze, Rainhard D Findling, and René Mayrhofer. A large-scale, long-term analysis of mobile device usage characteristics. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(2):1–21, 2017.

[LLL13] Kaikai Liu, Xinxin Liu, and Xiaolin Li. Guoguo: Enabling fine-grained indoor localization via smartphone. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 235–248. ACM, 2013.

[mat] Products and Services - MATLAB & Simulink. `https://www.mathworks.com/products.html`. Accessed: 2020-12-12.

[MNV03] Enzo Mumolo, Massimiliano Nolich, and Gianni Vercelli. Algorithms for acoustic localization based on microphone array in service robotics. *Robotics and Autonomous systems*, 42(2):69–88, 2003.

[Pah14] Marc-Oliver Pahl. *Distributed Smart Space Orchestration*. Dissertation, Technische Universität München, München, 2014.

[PD18] Anachack Phongtraychack and Darya Dolgaya. Evolution of mobile applications. In *MATEC Web of Conferences*, volume 155, page 01027. EDP Sciences, 2018.

[PL19] Marc-Oliver Pahl and Stefan Liebald. A Modular Distributed IoT Service Discovery. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 448–454. IEEE, 2019.

[plu] Plus Codes. `https://maps.google.com/pluscodes/`. Accessed: 2020-08-02.

[PP+16] Keyur K Patel, Sunil M Patel, et al. Internet of Things - IOT: definition, characteristics, architecture, enabling technologies, application & future challenges. *International journal of engineering science and computing*, 6(5), 2016.

[RSA+13] Mirco Rossi, Julia Seiter, Oliver Amft, Seraina Buchmeier, and Gerhard Tröster. RoomSense: an indoor positioning system for smartphones using active sound probing. In *Proceedings of the 4th Augmented Human International Conference*, pages 89–95. ACM, 2013.

[SW09] Stefan Saroiu and Alec Wolman. Enabling new mobile applications with location proofs. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, page 3. ACM, 2009.

[VBCMV+12] Mario Vega-Barbas, Diego Casado-Mansilla, Miguel A Valero, Diego López-de Ipina, José Bravo, and Francisco Flórez. Smart spaces and smart objects interoperability architecture (S3OiA). In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 725–730. IEEE, 2012.

[wha] what3words /// the simplest way to talk about location. `https://what3words.com/`. Accessed: 2020-08-02.

[WYM18] Xuyu Wang, Zhitao Yu, and Shiwen Mao. DeepML: Deep LSTM for indoor localization with smartphone magnetic and light sensors. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[ZC11] Zhichao Zhu and Guohong Cao. APPLAUS: A privacy-preserving location proof updating system for location-based services. In *2011 Proceedings IEEE INFOCOM*, pages 1889–1897. IEEE, 2011.

[ZGL19] Faheem Zafari, Athanasios Gkelias, and Kin K Leung. A survey of indoor localization systems and technologies. *IEEE Communications Surveys & Tutorials*, 2019.

[ZP15] Faheem Zafari and Ioannis Papapanagiotou. Enhancing ibeacon based microlocation with particle filtering. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2015.