University of Zurich^UZH

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

BluePIL: Fully Passive Identification and Localization of Bluetooth Devices in Near-Real-Time

Cyrill Halter
Zurich, Switzerland
Student ID: 13-928-171

Supervisor: Bruno Rodrigues, Christian Killer, Prof. Dr. Burkhard Stiller, Prof. Dr. Thomas Bocek
Date of Submission: 24.08.2020

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Zusammenfassung

Die strategische Planung und Evaluation von Marketingaktivitäten stützt sich auf eine genaue Bewertung des generierten Interesses an einem Produkt oder einer Dienstleistung. In öffentlichen Räumen wie Messen oder Verkaufsstellen bieten drahtlose Technologien die Möglichkeit, Leistungsindikatoren durch passiv abgefangene Funksignale abzuschätzen. Die Anzahl bluetoothfähiger Geräte hat über die vergangenen Jahre ein stetes Wachstum erfahren und soll bis 2024 auf 6.2 Mrd. steigen. Somit stellt Bluetooth für diese Aufgabe eine Alternative zu 802.11 Technologien dar und bietet Vorteile wie das Fehlen von MAC-Randomisierungsverfahren. In dieser Arbeit wird *BluePIL*, ein System zur Identifizierung und Lokalisierung von Bluetooth Geräten, beschrieben. Es unterscheidet sich durch ein vollständig passives Vorgehen von bestehenden Ansätzen, benötigt also keine Kenntnis oder Beteiligung von den Zielgeräten. Für die Identifizierung von Geräten verwendet das System Teile der Bluetooth-Adresse, für die Lokalisierung einen modifizierten Multilaterations-Algorithmus und ein Pfadverlust-Modell. *BluePIL* baut auf einer verteilten Streaming-Architektur auf und berechnet Ergebnisse in Fast-Echtzeit. Zur Evaluation des Systems wurde ein Prototyp in Python erstellt. Drei Experimente dienen als Grundlage für die Bewertung. Aus den Resultaten lässt sich eine Lokalisierungspräzision von ca. $1\,\mathrm{m}$ auf einer Fläche von $12\,\mathrm{m}^2$ und ca. $1.4\,\mathrm{m}$ auf einer Fläche von $25\,\mathrm{m}^2$ feststellen. Das System ist in der Lage, die spärlichen und verrauschten Daten des Sensors zu verarbeiten, und läuft ohne Probleme auf kostengünstiger Hardware. Insgesamt erfüllt das System die Anforderungen, welche für diese Arbeit definiert wurden. Durch Messungen an einer Konferenz wurde festgestellt, dass das Vorkommen von detektierbaren Bluetooth-Verbindungen in diesem Umfeld zu gering war, um die Anzahl Personen in einem Raum zuverlässig ableiten zu können. Dies zeigt auf, dass die Anwendbarkeit einer passiven Bluetooth-Lösung von dem Nutzungsgrad der Technologie abhängig ist.

Abstract

Measuring the interest in a product or service in a public space, such as a trade show or a sales floor, is fundamentally important for the evaluation of the marketing strategies of businesses. Wireless technologies provide a method of gauging performance indicators through the passive capture of signals emitted by mobile devices. Bluetooth technology has seen steady growth over the past years and continues to spread with the number of Bluetooth enabled devices expected to grow to 6.2B by 2024. Therefore, it provides a viable alternative to 802.11 wireless technologies in the accomplishment of this task and even offers advantages, such as the lack of MAC-randomization strategies. This thesis introduces *BluePIL*, a system for Bluetooth device identification and localization. It sets itself apart from existing approaches through its fully passive nature, where no collaboration with or prior knowledge of the devices to be tracked is necessary. The system relies on parts of the Bluetooth address for device identification and a modified multilateration algorithm using a path loss model for device localization. *BluePIL* is designed as a distributed streaming architecture that delivers results in near-real-time. A Python prototype serves as a basis for the evaluation of the system, which is accomplished through three experiments. The results show that the approach achieves good localization accuracies of $1\,\mathrm{m}$ to $1.4\,\mathrm{m}$ within a space of around $12\,\mathrm{m}^2$ and $25\,\mathrm{m}^2$ respectively. The system architecture is found to perform well regarding the sparse and noisy data received from the sensors and the available resources on the low-cost hardware used. Overall, the system designed satisfies the requirements defined for this thesis. Measurements performed at a conference reveal that the prevalence of detectable Bluetooth connections was not high enough to determine the number of people located within a defined space reliably. This shows that the applicability of a passive Bluetooth approach is dependent on the rate of utilization of the technology.

Acknowledgments

I would first like to thank my supervisor, Bruno Rodrigues, for his guidance and encouragement during the course of this thesis, for always taking the time to help it along and for providing direction where it was needed. I would also like to thank Prof. Dr. Burkhard Stiller for his support, for providing the opportunity to work on an interesting topic and for his assistance with organizational problems in the context of the SARS-CoV-2 pandemic, Simon Tuck for his insights into the Livealytics system and for his support during the experiment at the Reloading Live conference, and Lenz Baumann for the pleasant collaboration during joint experiments. Finally, I would like to thank Laura Schnoz for proofreading this thesis, for her valuable comments and for having an open ear for the issues encountered during the process of writing it.

Contents

Chapter 1

Introduction

1.1 Motivation

Measuring the public interest in a particular product or service is of fundamental importance for the strategic planning of businesses. In an increasingly digitized society, there are secure and non-invasive methods of visualizing an audience's mobility by passively tracking wireless signals emitted by portable devices. The analysis of passively emitted signals enables the extraction of key performance indicators (KPI) for the efficient planning of marketing strategies of business events or campaigns. This improves the offering of a product or service to a particular type of audience. In this regard, it is possible to obtain insights on how long visitors are engaged in an exhibition or how many visitors and observers are on site.

The major technical challenges within passive wireless tracking are the unique device identification and the correlation of devices to their users considering that users might carry more than one device emitting wireless signals, and these devices can emit passive wireless signals through different sources. Thus, it is possible to correlate the spatial and temporal dimensions through the received signal strength indication (RSSI) captured from different sources (Wi-Fi and Bluetooth) measured in a given point in time. This allows to determine the uniqueness of tracked devices with a higher likelihood. The major benefit of solving these challenges is to increase the precision of the calculated performance metrics based on accurate measurements.

The capture of signals can be based on wireless protocols, such as 802.11b or 802.11g (2.4 GHz) or 802.11a (5 GHz), but there is also the possibility of broadening the scope of signal frames, based on Bluetooth protocols. Ever since the introduction of the first Bluetooth protocol version in 1999, the technology has seen steady growth with the number of Bluetooth capable devices shipped expected to grow to 6.2B by 2024 [1]. The emergence of wearable devices, the growth of the internet of things (IoT) sector and the growing popularity of wireless audio devices have all contributed to the proliferation of Bluetooth in the wireless technology market.

The topic for this thesis emerged out of the InnoSuisse-funded cooperation between the Communication Systems Group at the University of Zurich and Livealytics, a Swiss company that offers an "analytics solution for measuring and benchmarking the performance of sales and live experience promotion activities, trade shows and retail spaces" [2] [3]. The Livealytics solution uses passively measured Wi-Fi signals to collect marketing statistics and KPIs at trade fairs, sales floors, etc. In practice, this consists of defining proximity zones around a Wi-Fi sensor and dwell times within these proximity zones that allow to classify visitors as having seen, visited or passed by a certain exhibit [2].

Technology-wise, this relies on Wi-Fi signal strengths and Wi-Fi media access control (MAC) addresses to calculate distances and identify individual devices respectively. The emergence of MAC-randomization strategies within Wi-Fi technology, however, has complicated the step of device identification. The fact that MAC-randomization does not exist in classic Bluetooth has spawned the hypothesis that Bluetooth may provide a viable alternative to the Wi-Fi approach. Having said this, the applicability of Bluetooth device tracking systems is not limited to the Livealytics use case. Most recently, for example, contact tracing applications for disease control have gained public interest in the context of the SARS-CoV-2 outbreak. Most of them rely on Bluetooth technology for proximity detection.

While a fair amount of research exists for both the area of unique identification and localization of Bluetooth devices, most approaches up to now do not allow a system to be fully passive, *i.e.* not requiring any knowledge of and collaboration with the target devices. This thesis will explore the possibilites of accomplishing this task. In addition to this, the Livealytics context brings with it a range of additional interesting design considerations, such as ease of use, low complexity in setup and configuration, and a distributed and cloud-friendly implementation.

1.2 Goals

This thesis aims to explore the possibilities and potential of Bluetooth technologies to implement a fully passive identification and localization approach for individuals using mobile devices. The main goals are twofold:

- **Research**: explore the scope of Bluetooth passive wireless capture strategies to provide the localization and an increased accuracy in the unique identification of devices by combining available parameters.

- **Engineering**: develop a Bluetooth sensing approach for identification and localization based on passively emitted signals as a proof-of-concept. The thesis should produce a working prototype, which can be interfaced with other modules and produces results to be contrasted with the thesis goals.

1.3 Methodology

This thesis involves two main stages, organized as research and engineering. The first stage involves the identification of strategies in existing research for the unique identification and the localization of Bluetooth devices, in order for them to be tracked uniquely. This stage has an exploratory character. The second stage requires to put into practice said researched elements, in order to fulfill the second goal and build the proof-of-concept able to sense Bluetooth devices. Further, this stage involves the evaluation the solution to verify whether it is capable of satisfying the goal.

1.4 Thesis Outline

The rest of this thesis is structured as follows: Chapter 2 explains the basic knowledge required for the understanding of the concepts in this thesis and describes existing work related to the topic. Chapter 3 introduces the *BluePIL* system including the requirements to be fulfilled, the overall system architecture and the individual components developed for this thesis. Chapter 4 describes the concrete implementation of *BluePIL* as a Python application. The system is then evaluated in Chapter 5 in three separate experiments. Finally, Chapter 6 concludes this thesis and list possible directions for future research.

Chapter 2

Background and Related Work

2.1 Background

2.1.1 Bluetooth

Bluetooth is a ”short-range communications system intended to replace the cable(s) connecting portable and/or fixed electronic devices” [4]. Operating in the unlicensed 2.4 GHz industrial, scientific and medical (ISM) frequency band, Bluetooth devices typically transmit up to a distance of 10 m to serve this purpose. Typical uses for Bluetooth technology include audio streaming, *e.g.* for wireless headphones, data transfer, *e.g.* for the communication with wearable devices, location services, *e.g.* for asset tracking in supply chain management solutions, and the construction of device networks, *e.g.* for smart home products [1].

The Bluetooth Core Specification is composed by the Bluetooth Special Interest Group (SIG) and is currently at Version 5.2 [4]. The Bluetooth SIG is a not-for-profit standards organization that ”expands Bluetooth technology by fostering member collaboration to create new and improved specifications, drives global Bluetooth interoperability through a world class product qualification program, and grows the Bluetooth brand by increasing the awareness, understanding, and adoption of Bluetooth technology” [5].

Since Core Specification Version 4.0, Bluetooth has been split into two separate protocol stacks [4]: Bluetooth Basic Rate / Enhanced Data Rate (BTBR/EDR), sometimes called *Classic Bluetooth*, was developed out of the original Bluetooth version published in Core Specification Version 1.0. It maintains backwards compatibility to legacy protocol versions. Bluetooth Low Energy (BTLE) was newly introduced with Version 4.0. It implements a completely separate protocol stack and is not compatible to BTBR/EDR devices. BTLE was created to cater to the energy efficiency requirements, the complexity limitations and the cost constraints of modern IoT devices. These two protocol stacks will be described in more detail in Subsection 2.1.2 and 2.1.3 respectively.

Generally speaking, both the BTBR/EDR and the BTLE stacks are separated into a controller, *i.e.* the part of the stack running on a controller module, and a host component,

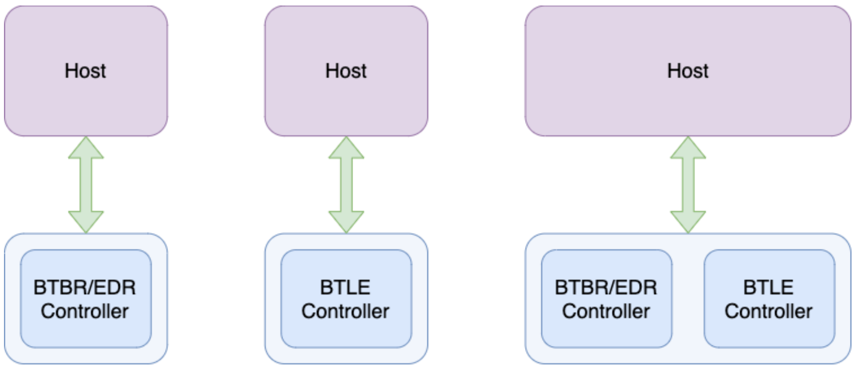


Figure 2.1: Possible combinations of BTBR/EDR and BTLE controllers [4]

Device Class	**Max. Transmission Power**	**Max. Transmission Distance**
Class 3	0 dBm	ca. 10 cm
Class 2	4 dBm	ca. 10 m
Class 1	20 dBm	ca. 100 m

Table 2.1: Classes of BTBR/EDR devices [4]

i.e. the part of the stack running on the host system. The two are connected through the Host Controller Interface (HCI). A Bluetooth module may include a BTBR/EDR controller, a BTLE controller, or both, as is shown in Figure 2.1. For the purposes of this thesis, we will only be focusing on the controller component and ignore the upper parts of the respective protocol stacks.

2.1.2 Bluetooth Basic Rate / Enhanced Data Rate (BTBR/EDR)

This subsection will introduce the parts of the BTBR/EDR protocol that are relevant to this thesis, as they are described in the Bluetooth Core Specification v5.2 [4].

BTBR/EDR is a protocol stack described in the Bluetooth Core Specification [4]. The BTBR/EDR controller component of the protocol stack consists of the Radio Layer, the Baseband Layer and the Link Manager Layer. For this thesis, we will focus on the Radio and the Baseband Layer, since they are generic to all communication running over BTBR/EDR.

The Radio Layer describes the characteristics required of the radio device running the transmission [4]. BTBR/EDR operates in the 2.4 GHz unlicensed ISM band. It uses a Frequency Hopping Spread Spectrum (FHSS) scheme over 79 distinct channels spread across the spectrum to avoid interference and fading, hopping at a rate of $1'600$ hops/s. It uses Gaussian Frequency-Shift Keying (GFSK) modulation to encode the binary message to be transmitted. BTBR/EDR devices are split into three device classes according to their maximum transmit power level (*cf.* Table 2.1), class 2 being the most wide-spread.

The Baseband Layer describes the physical links used to enable point-to-point and point-to-multipoint connections between devices [4]. A BTBR/EDR physical channel may be

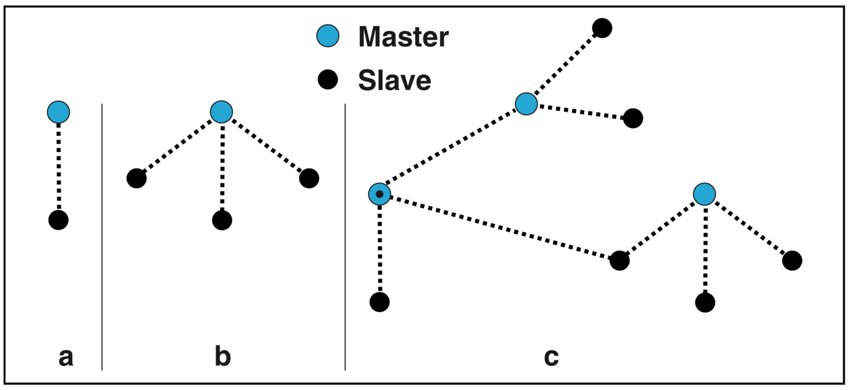


Figure 2.2: Possible piconet topologies in BTBR/EDR: a point-to-point connection **(a)**, a point-to-multipoint connection **(b)** and a scatternet **(c)** [4].

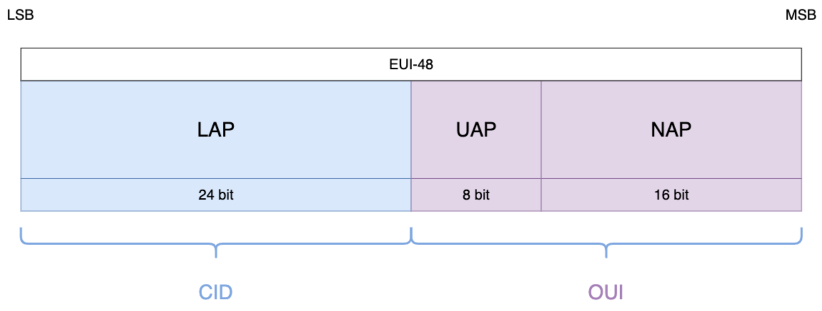


Figure 2.3: The composition of the Bluetooth address [4]

shared by two to eight devices connected in a so-called *piconet*. A piconet is characterized by a shared hopping pattern and clock, a *master* node and one to seven *slave* nodes. Devices may participate in multiple piconets simultaneously through Time Division Duplex (TDD), forming a so-called *scatternet*. These possible piconet topologies are summarized in Figure 2.2. The master node defines the piconet's clock and hopping pattern, a pseudorandom ordering of the 79 available channels. An additional physical channel exists for inquiry scans. This channel is used to discover devices that are in a connectable state. It follows a slower hopping pattern that is common to all BTBR/EDR devices.

To account for interference and environmental factors, connected Bluetooth devices may dynamically adjust the hopping pattern used to exclude certain frequencies where there is a lot of interference [4]. It is also possible to dynamically increase or decrease the transmit power level depending on the measured signal strength at the receiving device.

Every Bluetooth device has a unique Bluetooth address [4]. This address is constructed as a 48-bit Extended Unique Identifier (EUI-48) according to the IEEE Standard for Local and Metropolitan Area Networks [6]. It is composed of a 24-bit company ID (CID) and a 24-bit organizationally unique ID (OUI). The CID is vendor-assigned, whereas the OUI has to be obtained from the IEEE Registrational Authority and is assigned to individual organizations, manufacturers or vendors of Bluetooth technology. Within the domain of Bluetooth networking, the parts of the Bluetooth address are differentiated further by the lower address part (LAP), corresponding to the CID, and the 16-bit non-significant address part and 8-bit upper address part, forming the OUI. This is illustrated in Figure 2.3.

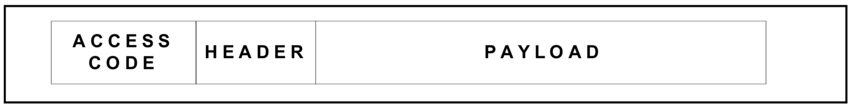


Figure 2.4: The generic structure of a BTBR packet [4]

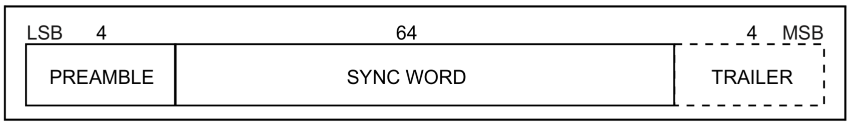


Figure 2.5: The composition of the BTBR access code [4]

Similar to other networking protocols, Bluetooth encapsulates data in packets [4]. Figure 2.4 shows the generic composition of a BTBR packet. The composition of the access code (AC) is shown in further detail in Figure 2.5. The syncword is always derived from the LAP of a specific device involved in the transmission. Which device is chosen depends on the type of packet being transmitted. For the case of an ongoing data transmission in a piconet, the channel access code (CAC) is used, which is derived from the master device's LAP. Furthermore, it includes information for clock synchronization as well as error correction data.

2.1.3 Bluetooth Low Energy (BTLE)

This subsection will introduce the parts of the BTLE protocol that are relevant to this thesis. BTLE is a protocol stack described in the Bluetooth Core Specification [4]. The BTLE controller component of the protocol stack consists of the LE PHY Layer, and the Link Layer.

The LE PHY Layer describes the characteristics required of the radio device running the transmission [4]. BTLE operates in the 2.4 GHz unlicensed ISM band. Similar to BTBR/EDR, BTLE uses a FHSS scheme to avoid interference and fading, the difference being that only 37 channels are used. Three additional channels exist for advertising. BTLE uses GFSK to encode the message to be sent. Similar to BTBR/EDR, different device classes exist, corresponding to the maximum transmit power level (*cf.* Table 2.2).

The Link Layer describes the physical links used to enable point-to-point and point-to-multipoint connections between devices [4]. It is where BTLE manages to generate the power savings that set it apart from BTBR/EDR. Different to a piconet in classic

Device Class	**Max. Transmission Power**	**Max. Transmission Distance**
Class 3	0 dBm	ca. 10 cm
Class 2	4 dBm	ca. 10 m
Class 1.5	10 dBm	ca. 30 m
Class 1	20 dBm	ca. 100 m

Table 2.2: Classes of BTLE devices [4]

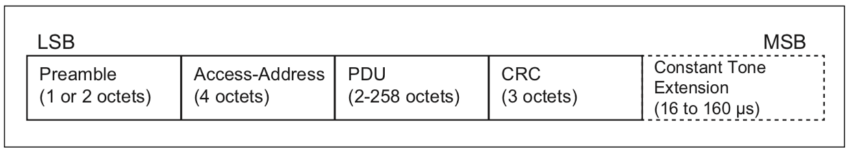

LSB
MSB
Preamble (1 or 2 octets)
Access-Address (4 octets)
PDU (2-258 octets)
CRC (3 octets)
Constant Tone Extension (16 to 160 µs)

Figure 2.6: The generic structure of a BTLE Uncoded PHY packet [7]

Bluetooth, devices don't all share a common physical channel in a BTLE piconet. Instead, each slave negotiates a separate hopping pattern with the master device. This avoids the slaves having to constantly listen for incoming connections and allows them to actively request data themselves, relocating the decision when to expend power to the slave nodes. This also means that the possible number of devices in a BTLE piconet is only limited by the resources of the master device. The hopping pattern used in a BTLE physical channel is determined through the advertising packet sent by a connection initiating device.

Figure 2.6 shows the generic structure of a BTLE Uncoded PHY packet. The Access-Address (AA) corresponds to the AC in a BTBR packet in function. Other than the BTBR version, however, it is not derived from the LAP of the Bluetooth address. Instead, in BTLE, the AA is randomly generated.

2.1.4 Project Ubertooth

Project Ubertooth is a fully open-source hardware and software package for wireless development [7]. It is suitable for experimentation with both BTLE and BTBR/EDR. The Ubertooth allows access to the lower layers of the Bluetooth protocols, which are normally hidden in off-the-shelf Bluetooth modules. It does this at a low cost, being available for purchase at around 100$. This is especially remarkable since the price quickly skyrockets for alternatives that can be found on the market. The *Ellisys Bluetooth Explorer 400*, for example, sells at around 20'000$ for a complete system [8].

Project Ubertooth manages to keep the overall costs low thanks to one main simplification over other devices on the market. Bluetooth is a spread spectrum technology, meaning that it moves over a wide range of frequencies during the transmission of data. Most Bluetooth monitoring hardware, therefore, implements an array of transceivers to observe all channels used by the Bluetooth protocol simultaneously. The Ubertooth One, however, only uses a single transceiver, opting instead to try and hop along with the hopping pattern of ongoing Bluetooth connections in order to eavesdrop on the data being transmitted [7]. This leads to a fairly capable device at a price that makes it an ideal option for non-commercial applications, such as academia.

The Ubertooth One is built around three main hardware components [7]: The *Texas Instruments (TI) CC2591 RF Front End*, the *TI CC2400 Wireless Transceiver* and an *NXP LPC175x Series ARM Cortex-M3 Microcontroller* with full-speed USB 2.0 (*cf.* Figure 2.7). The analog signal is prepared by the CC2591 for the CC2400, capable of working in the 2.4 GHz spectrum and detuning at the rate of $1'600$ hops/s necessary for working with Bluetooth. It is then processed on the LPC175x microprocessor, which finally transmits it to the host system. The Ubertooth One hardware setup, however, leads to some

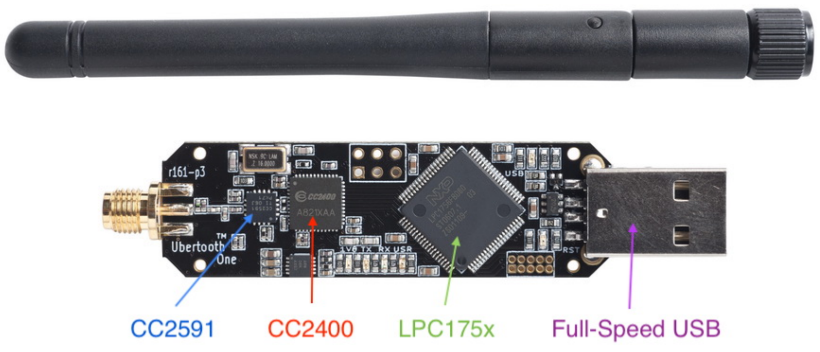


Figure 2.7: The composition of the Ubertooth One device [9]

trade-offs. The CC2400 can receive at a maximum of 1 Mbit/s. This makes it unable to process BTEDR fully, which can reach maxima of 2.1 Mbit/s [8]. Consequently, only BTBR is fully supported by the Ubertooth One.

The Ubertooth software package is written in C [7]. It is split up into two parts. The Ubertooth Firmware code runs on the LPX175x microprocessor and is responsible for controlling the CC2400 transceiver, processing the data received from the CC2400 and sending said processed data to the host via USB. This is especially relevant for time-sensitive operations, such as the synchronization to a piconet clock, which does not have to make the detour to the host system and back via USB thanks to the LPC175x. The Ubertooth Host code runs on on the host system, *i.e.* any computer with a USB 2.0 port and capable of running the compiled Ubertooth host code. It sends high-level control signals to the Ubertooth device, receives data from it and converts it into a format that is useful to the end user, *e.g.* as a PCAP file. The Ubertooth Host code is paired with *libbtbb*, a Bluetooth Baseband decoding library developed in conjunction with Project Ubertooth. As an open-source project, the Ubertooth software package is under constant development and remains incomplete in some aspects.

2.1.5 Kalman Filtering

The Kalman Filter describes a type of Bayesian filter. Bayesian filters ”probabilistically estimate a dynamic system’s state from noisy observations” [10]. With the state at time t represented as random variable s_t, it can be expressed mathematically as finding the probability distribution over s_t, which we then call *belief* $Bel(s_t)$. With sensor observations over time $z_1, z_t, ..., z_t$, the belief is defined as follows:

$$Bel(s_t) = p(s_t|z_1, z_2, ..., z_t) \tag{2.1}$$

Bayes’ theorem tells us that this belief can also be expressed in the following way [11]:

$$p(s_t|z_1, z_2, ..., z_t) = p(s_t|z) = \frac{p(z|s_t) * p(s_t)}{p(z)} \tag{2.2}$$

$p(z|s_t)$ is the probability of making the observations z, given that we are in state s_t. $p(s_t)$ is the probability of being in the state s_t prior to our knowledge of the observations z. $p(z)$, finally, is the probability of observing z without any restrictions by the state s_t. This theorem is often expressed in the more high-level way

$$\text{posterior} = \frac{\text{likelihood} * \text{prior}}{\text{evidence}} \tag{2.3}$$

The Kalman Filter is then nothing more than a Bayesian filter that assumes the probability distributions to be Gaussian. It will be explained in the following [11] [12].

With

s_k The state at time k

z_k The observation at time k

F_k The state transition model at time k

H_k The observation model at time k

Q_k The process noise covariance at time k

R_k The observation noise covariance at time k

we can define the state at time k as

$$s_k = F_k s_{k-1} + w_k, w_k \sim \mathcal{N}(0, Q_k) \tag{2.4}$$

and the observation at time k

$$z_k = H_k s_k + v_k, v_k \sim \mathcal{N}(0, R_k) \tag{2.5}$$

The Kalman Filter Algorithm

We can now define the Kalman Filter algorithm as an iterative algorithm in two steps: the predict step and the update step [12] [11]. In the following, the hat operator "$\wedge$" will denote an estimate of a variable, the superscript "$-$" will denote a predicted (prior) and the superscript "+" an updated (posterior) estimate.

During the **Predict Step**, we calculate the *predicted state estimate* $\hat{s}_k^-$ and the *predicted error covariance* P_k^-

$$\hat{s}_k^- = F_k s_{k-1}^+ \tag{2.6}$$

$$P_K^- = F_k P_{k-1}^+ F_k^T + Q_k \tag{2.7}$$

During the **Update Step**, we correct the predictions using our measurements. We first calculate the *measurement residual* $\tilde{y}_k$ and its covariance S_k.

$$\tilde{y}_k = z_k - H_k \hat{s}_k^- \tag{2.8}$$

$$S_k^- = H_k P_k^- H_k^T + R_k \tag{2.9}$$

We can then calculate the *Kalman gain* K_k.

$$K_k = P_k^- H_k^T S_k^{-1} \tag{2.10}$$

Finally, we can compute the *updated state estimate* $\hat{s}_k^+$ and the *updated error covariance* P_k^+

$$\hat{s}_k^+ = \hat{s}_k^- + K_k \tilde{y}_k \tag{2.11}$$

$$P_k^+ = (I - K_k H_k) P_k^- \tag{2.12}$$

2.1.6 Multilateration

Multilateration is the process of geometrically estimating an object's position in space through distance measures to at least three points. For the case where exactly three points are used, we call it trilateration. Mathematically, this corresponds to solving the following non-linear system, with (x_i, y_i, z_i) the position of the i-th point, (x, y, z) the position of the object, and d_i the distance of the object to the i-th point.

$$\begin{aligned} (x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 &= d_1^2 \\ (x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 &= d_2^2 \\ (x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 &= d_3^2 \end{aligned} \tag{2.13}$$

For planar problems, this can be simplified further, as is illustrated in Figure 2.8. This leads to the following system in two variables.

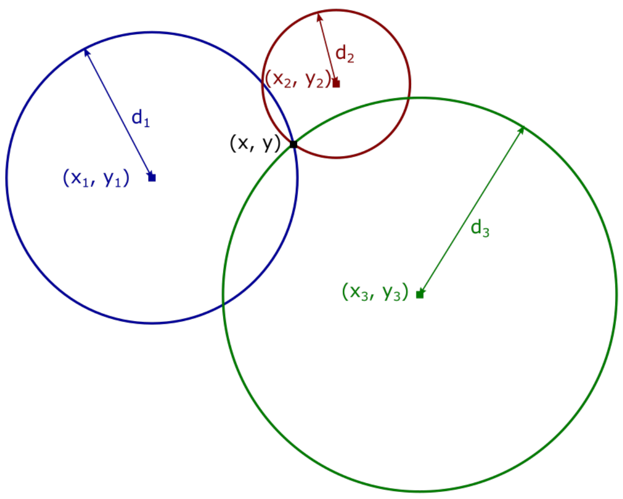


Figure 2.8: A planar trilateration problem

$$\begin{aligned}(x - x_1)^2 + (y - y_1)^2 &= d_1^2 \\ (x - x_2)^2 + (y - y_2)^2 &= d_2^2 \\ (x - x_3)^2 + (y - y_3)^2 &= d_3^2\end{aligned} \tag{2.14}$$

This system is then often linearized by subtracting the last equation from the other two, leading to the following determined linear system of equations [13].

$$\begin{aligned}2(x_3 - x_1)x + 2(y_3 - y_1)y &= d_1^2 - d_3^2 + x_3^2 - x_1^2 + y_3^2 - y_1^2 \\ 2(x_3 - x_2)x + 2(y_3 - y_2)y &= d_2^2 - d_3^2 + x_3^2 - x_2^2 + y_3^2 - y_2^2\end{aligned} \tag{2.15}$$

The solution is reached analogously for any higher order multilateration problems. In practice, distance measures are often imperfect and the calculation of a solution for Equation 2.14 using a non-linear optimization lead to better results.

2.1.7 The Log-Distance Path Loss Model

The Log-Distance Path Loss Model is a popular model for radio signal decay over distance [14]. It models the finding that the decay of a signal over distance can be approximated by a logarithmic function. With $RSS(d)$ the received signal strength at distance d, d_0

Source	Technology Used	Nature	Method of Device Identification	Notes
[15]	BTLE	Passive	Identifies the AA from passively captured empty data packets	AA does not identify a device and may change over time
[16]	BTLE	Passive	Tracks devices through AA and carry-over algorithm using identifying tokens in advertising packets	Results heavily dependent on smartphone OS
[17]	BTLE	Passive	Fingerprints devices through mobile app UUIDs revealed in advertising packages	Heavily tailored to Android devices
[18]	BTBR/EDR	Passive	Tracks devices through fingerprinting algorithm using the clock skew	Robust to manual modification of Bluetooth packet contents, *s.a.* MAC randomization but requires modification to off-the-shelf hardware
[19]	BTBR/EDR	Active	Circumvention of non-discoverability through targeted inquiry requests	Vulnerability may no longer be current and the exploit is computationally very expensive
[20]	BTBR/EDR	Semi-Passive	Fingerprinting through various device characteristics from advertising packets	Requires target device to be in discoverable mode
[21]	BTBR/EDR	Passive	Discovery of the LAP from the Bluetooth packet AC, UAP through the reversal of the the computation of the Header Error Code (HEC)	Special hardware required, implemented in Project Ubertooth [7]

Table 2.3: Related work on Bluetooth device identification

a reference distance, n the path-loss coefficient and $\mathcal{X}_\sigma$ a zero-mean Gaussian random variable, it can be defined as follows:

$$RSS(d) = RSS(d_0) - 10n\log\left(\frac{d}{d_0}\right) + \mathcal{X}_\sigma \tag{2.16}$$

In practice, the reference distance d_0 is often set to 1 m and noise is ignored for the calculation, simplifying the model even further. With RSS_C, the received signal strength at 1 m, it can then be expressed as follows:

$$RSS(d) = RSS_C - 10n\log(d) \tag{2.17}$$

RSS_C is dependent on each individual device and has to be calibrated for. The path-loss coefficient n is a factor that depends on the environment. For free-space, it is often chosen at $n = 2$.

2.2 Related Work

2.2.1 Device Identification

The following subsection will summarize existing work on Bluetooth device identification. A brief comparison is presented in Table 2.3. Generally speaking, the approaches can be categorized into research that uses BTLE and that which uses BTBR/EDR technology.

In [15], the BTLE AA is extracted from empty data packets that are captured passively. While the AA does carry some identifying information regarding the link between two devices, it does not identify a device as such and may change over time. [16] builds upon this by tracking AA changes over time, using other fields in BTLE packages to implement a carry-over algorithm. This improves the result but is heavily dependent on the OS running on devices that participate in the communication. [17] takes a completely different approach, relying on universally unique identifiers (UUIDs) written into BTLE advertising packets by mobile apps in order to create a fingerprint for a specific device. This approach is tailored mostly to *Android* devices and fairly computationally complex, requiring the Android application packages (APKs) to be scanned for vulnerabilities before being able to extract fingerprints from running connections.

In the BTBR/EDR domain, [18] approaches the device identification from the physical angle, fingerprinting devices based on the rate of drift of packet inter-arrival times caused by the master device's clock skew. The fact that this approach is implemented on the physical level means that it is robust to any security measures implemented in upper layers of the BTBR/EDR protocol, such as non-discoverability or MAC randomization. However, it also requires modifications to standard off-the-shelf BTBR/EDR hardware in order to work. [19] tries to circumvent the non-discoverability of devices by actively sending targeted inquiry requests to all devices in a certain address range in a brute-force manner. It exploits a vulnerability where these devices will then respond to the requests, even though they should not be discoverable. This vulnerability is no longer current in all devices and the exploit is computationally very expensive since it requires an exhaustive traversal of the address space in question. [20] fingerprints a device through information that is retrieved from passively captured advertising packets. This approach only works if a device is discoverable, which is not the case per default in most current Bluetooth implementations. Finally, [21] uses specialized hardware to extract parts of the Bluetooth address from an ongoing BTBR/EDR connection. The LAP is obtained from the syncword of the Bluetooth packet AC, the UAP by reversing the computation of the Header Error Code (HEC). The only part of the Bluetooth address that this method fails to obtain is the NAP, since it is not integrated in any way into the meta information of a Bluetooth packet.

2.2.2 Device Localization

The following subsection will summarize existing work on Bluetooth device localization. A brief comparison is presented in Table 2.4. The approaches explored are all active in nature, *i.e.* they require knowledge of or collaboration with the device to be localized. On a high level, they can be divided into fingerprinting-based and path-loss-model-based methods.

The fingerprinting-based approaches ([22] [23]) require the construction of a radio map of the area of interest, *i.e.* a set of sensor measurements for signal strength throughout the space, as a preliminary step. During localization, the signal strength measured for a mobile device within this space is then correlated with the individual points in the radio map. The best match is then assumed to correspond to the actual location of the device. [22] uses a large number of BTLE beacons set up throughout a building. A mobile device then measures the signal strengths from these beacons in order to position itself within the space. The approach includes methods for the continuous tracking of devices where a localization result can be used to improve the estimation in subsequent measurements. The paper also includes a detailed evaluation of the key parameters in the positioning algorithm. [23] implements a BTBR/EDR-beacon-based fingerprinting

Source	Technology Used	Nature	Method of Device Localization	Localization Accuracy	Notes
[22]	BTLE	Active	Location fingerprinting using RSS values from BTLE Beacon advertising packets	< 2.6 m	Includes methods for one-shot localization and tracking
[23]	BTBR/EDR	Active	Location fingerprinting for BT-BR/EDR Beacon advertising packets	~ 1.6 m	Evaluates k-Nearest-Neighbors and Naïve Bayes classifiers for fingerprint correlation
[24]	BTLE	Active	Location fingerprinting and a polynomial regression path loss model, fused using a Kalman Filter	~ 1.65 m	Multiple layers of statistical outlier filtering included
[25]	BTLE	Active	Trilateration using the log-distance path loss model with Kalman Filter for signal pre-processing for BTLE Beacon advertising packets	~ 0.3 m	
[26]	BTLE	Active	Trilateration using the log-distance path loss model for BTLE Beacon advertising packets	~ 1 m	Fuses trilateration results for more than three sensors, includes strategies for space partitioning
[13]	BTLE	Active	Trilateration using a neural network (NN) based path loss model and a Kalman Filter for signal pre-processing for BTLE Beacon advertising packets	~ 0.7 m	Variations in the advertising packet signal strengths are dynamically accounted for using an in-range reference node
[27]	BTLE	Active	Log-distance path loss model or particle filter using BTLE Beacon advertising packets	~ 0.6 m	Reduce positioning problem to one dimension heuristically by enforcing a shopping mall topology
[28]	BTBR/EDR	Active	Trilateration using the log-distance path loss model for BTBR/EDR advertising packets	Not listed	Includes investigation of the influence of the human body on RSS values
[29]	BTBR/EDR	Active	Trilateration using the log-distance path loss model	n/a	Known positions and the localization results are used to estimate the channel parameters $RSS(d_0)$ and n

Table 2.4: Related work on Bluetooth device localization

approach. In addition, it compares the performance of k-Nearest-Neighbors and Naïve Bayes classifiers for fingerprint correlation.

The path-loss-based approaches ([25] [26] [13] [27] [28] [29]) rely on a model that enables them to convert signal strengths to distances. [25] implements a log-distance path loss model to calculate distances for BTLE beacon advertising packets, which are then used to perform trilateration in order to compute a location for the target device. In addition to this, they use a Kalman Filter to pre-process the signal strength measurements to get a smoother and more viable result. [26] takes a similar approach with a combination of the log-distance path loss model and trilateration. They extend this by fusing the trilateration results multiple combinations of three sensors in order to improve the localization accuracy. [13] also uses trilateration but implement a neural-network-based path loss model that uses empirical data to infer a mathematical representation of the path loss characteristics. The signal strengths are pre-processed using a Kalman filter. A method is also presented to deal with the problem of varying signal strengths in BTLE beacon advertising packets. To this end, a reference node is included in the localization space which allows the system to dynamically adjust. [27] compares the performance of the log-distance path loss model and a particle filter for the localization problem. The trilateration step is avoided by heuristically reducing the localization problem to a one-dimensional one through assumptions on the topology of the environment they are working in, *i.e.* a shopping mall. [28] uses BTBR/EDR advertising packets to perform localization using the log-distance path loss model and trilateration. This work includes the investigation of the influences of the human body on signal strength measurements. [27] also work with the log-distance path loss model and trilateration. The goal here, however, is not the calculation of a location. Instead, the location is assumed to be known and this information is used to estimate the channel parameters RSS_C and n.

Finally, [24] presents an approach that fuses fingerprinting and a polynomial regression path loss model. Distance estimates from the fingerprinting algorithm and the path loss model are averaged and fed to an extended Kalman filter, which performs the localization and smooths the result. Additionally, multiple layers of outlier detection are implemented in order to improve the overall localization accuracy.

2.2.3 Applications

This subsection will introduce some applications in existing research that implement Bluetooth device identification and localization technology. [30] describes an application that is used to mine visitations patterns at tourist attraction in the Belgian city of Ghent. They use Bluetooth to identify individual devices given that they are discoverable. [31] proposes a system for presence detection of individuals in smart homes in order to enable energy savings, assistance for elderly or impaired people and the personalization of the smart home experience. A set of Bluetooth beacons is used to enable the identification of individuals and the localization on room-level. [32] uses a combination of Bluetooth and Wi-Fi to estimate crowd densities and pedestrian flows at an airport. Repeated inquiry scans deliver information on the number of discoverable Bluetooth devices in the vicinity of the sensors. The recent SARS-CoV-2 outbreak has seen widespread interest in mobile applications for contact tracing. *DP-3T* is an example for such an application and has been deployed in Switzerland as the official SARS-CoV-2 contact tracing application by the health authorities [33]. It uses the *Exposure Notification API*, a joint specification by *Apple* and *Google*, to detect and notify close devices via BTLE [34].

Chapter 3

System Design

The following chapter introduces the *BluePIL* system, a fully passive approach to Bluetooth device identification and localization. The design considerations, required characteristics and system design are described on an abstract level. Implementation details are introduced at a later point providing a detailed description of an instantiation of *BluePIL*'s architecture.

3.1 Design Considerations

The system described in this chapter is designed within the context of the Livealytics solution, a service that uses passively measured Wi-Fi signals to collect marketing statistics and KPIs in public spaces [2]. While the applicability of the design is not limited to the Livealytics case, this provides interesting considerations to be taken into account for a system deployed in real-world situations, some of which will be described in the following:

Low-Performance Hardware: The hardware available may be limited in performance due to a number of reasons. Economic factors may put a cap on the amount of money that can be spent on a single hardware unit. In any case, using cheaper and, thus, lower-performance devices improves profit margins that can be gained from a product. Hardware may also be limited in certain physical characteristics, such as size or power consumption, introducing further constraints for performance.

Cloud-Based Environment: Many existing systems run in a cloud-based environment. This allows businesses to decrease the time-to-market and to scale the resources used to the current demand on the system. Cloud-based environments bring along with them a number of economic considerations. In general, it is beneficial to reduce the amount of data that needs to be stored in, processed by and passing through the cloud [35].

Real-Time Evaluation: Real-time data evaluation allows users to react to the events processed by a system as they are happening. This may provide an economic benefit and enables applications to be built that are reactive to the current situation. On a technological level, real-time data evaluation eliminates the need for the storage of intermediate results and may, therefore, save resources in the overall system.

Ease of Use: An easy to use system may bring along economic benefits. It can be marketed towards non-expert users and can, therefore, gain a larger user-base. A simple product may also be preferred by potential buyers over a complicated one. Furthermore, lower complexity in the utilization of a product will decrease the amount of errors that occur on the user side and will thereby increase the quality of the data produced by the system.

3.2 Requirements

From the goals of this thesis (*cf.* Section 1.2), we can derive the following main requirements on the system to be built:

R1 The system should be completely passive.

R2 The system should be able to identify individuals based on a Bluetooth signal.

R3 The system should be able to localize individuals based on a Bluetooth signal.

Furthermore, the following additional requirements are derived from the design considerations described in Section 3.1. These requirements are not imperative for the fulfillment of the project goals, but bring along benefits for the real-world applicability of the system to be built.

AR1 The system should be able to work with low-performance hardware.

AR2 The system should be suited for a cloud-based environment.

AR3 The system should produce results in real time.

AR4 The system should be easy to use and require a minimal amount of configuration.

3.3 Assumptions

Considering the development of the system within the scope of this thesis, whose time frame is limited, assumptions are required to simplify its development, deployment and operation towards a posterior evaluation. Thus, to define the scope of the requirements described in Section 3.2, the following assumptions are made regarding the environment the system will be working in:

- An existing BTBR/EDR is actively being used on the target device, *e.g.* an audio stream to a pair of Bluetooth headphones.
- The localization problem can be reduced to a planar space, *i.e.* a location only has to be calculated in two dimensions.

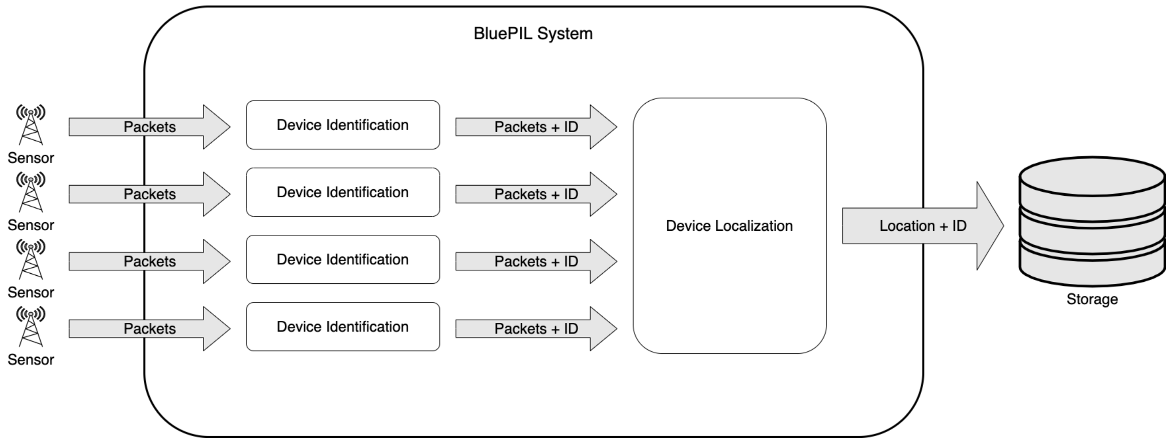


Figure 3.1: The *BluePIL* system topology on a high level

- The number of sensors available to the system is four.
- The sensors used have access to the lower layers of the BTBR/EDR protocol.
- The localization is performed in the area between the four sensors.
- The target device is a mobile device, such as a smartphone or a tablet.

3.4 High-Level System Architecture

The following describes the *BluePIL* system architecture, that uses passively captured BTBR/EDR signals based on four sensors to fulfill the requirements described in Section 3.2. The system topology was designed to make the approach flexible in deployment and in the concrete implementation of the individual components. In general, *BluePIL* is set up as a streaming data processing pipeline. Figure 3.1 shows this topology on a high level. Packets are delivered as a stream from a Bluetooth sensor. These packets first get processed by the device identification component, which are described in detail in Section 3.5. The packets along with their extracted identifiers then get passed along to the device localization component, which is explained in Section 3.6. It is itself a component composed of several sub-components, the topology of which is shown in Figure 3.3.

The design of the *BluePIL* system as a streaming multi-component architecture allows it to be flexible regarding the implementation of the individual components, *i.e.* the architecture itself is not invalidated by the choice of the implementation of a concrete component. It allows for the logical processing entities, be they physical or virtual, in a deployed system to be configured in different ways. Here, we chose a distributed node-sink setup, *i.e.* a system where many physical nodes send data to a single physical sink, which is then responsible for forwarding the data to an entity where it can be stored or processed (*e.g* the cloud). Computations are performed as early as possible to avoid bottlenecks downstream and to reduce the amount of data that has to be forwarded by the sink. The resulting topology is shown in Figure 3.2. Device identification and signal strength filtering are performed on individual nodes, moving computational effort away

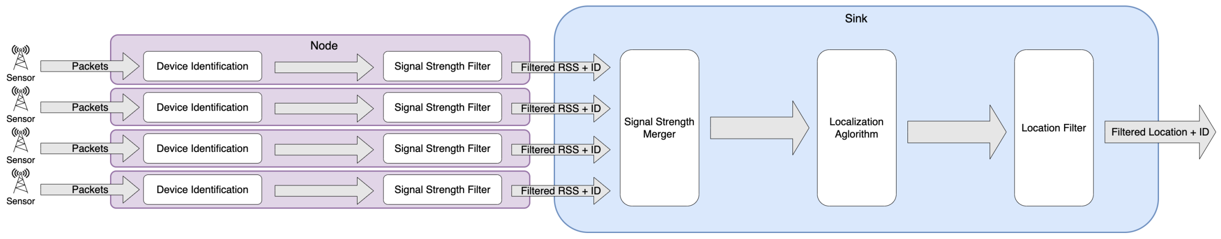


Figure 3.2: The chosen deployment for the *BluePIL* system

from a single sink. The merging of the individual signal streams necessitates a centralized entity and is performed on the sink, including any step that follows it. It is important to note that this is only one possible deployment of the *BluePIL* system. For example, the streaming architecture would permit any part of the processing pipeline to be performed on the cloud. This allows for economic considerations to be taken into account in the choice of a concrete deployment setup.

3.5 Device Identification

The approach described in [21] is implemented in this thesis. It allows the system to profit from the fact that BTBR/EDR lacks any sort of MAC randomization and avoids having to build a complex, low-level system for fingerprinting by using a unique identifier that is already available: the Bluetooth address. As is described in Subsection 2.1.2, the Bluetooth address consists of the LAP, the NAP, and the UAP. These three address parts and their usefulness for the device identification problem are discussed separately in the following.

The NAP is, as the name says, not significant, and can, therefore, not be derived from passively captured Bluetooth traffic. [21] describes an approach, where possible NAP values are selected heuristically from the list of all manufacturer OUIs. These values, however, have to be validated through a targeted inquiry request, which breaks with the passive nature of *BluePIL*.

The LAP is the most easily obtainable from passively captured Bluetooth traffic. It is contained in the CAC in any piconet packet, specifically in the syncword, and can simply be read without the need for any further processing [21] [4]. In addition, it can be validated using the 34-bit checksum included in the syncword, based on a (64, 30) expurgated block code [4].

The computation of the UAP is complex since it is only contained in a packet implicitly. This is complicated further by the fact that everything past the AC in the Bluetooth packet is whitened, *i.e.* scrambled by XORing it with a whitening word derived from six bits in the master Bluetooth clock [4]. [21] describes a method where candidate packets are produced by de-scrambling packets with all 64 possible whitening words. They then exploit the fact that the UAP is used to initialize the Linear-Feedback Shift Register (LFSR) for the calculation of the HEC. This operation is completely reversible and is used to generate 64 candidate UAPs, which are then validated by checking the consistency of certain packet header fields over multiple packets, and by checking the payload Cyclic Redundancy Check (CRC).

While the method for the derivation of the UAP may produce satisfactory results, it is computationally fairly expensive and may take time due to the fact that multiple packets are necessary for the validation of the candidate packets. This may cause a problem in the environment at hand, which is constrained in both computational resources (low-cost sensors) and time (real-time analysis). The following heuristic is, therefore, used for this thesis: The LAP, while not globally unique, is unique enough to identify devices under certain circumstances. As is stated in Section 3.3, this thesis deals with mobile devices, such as smartphones, tablets, etc. The five biggest smartphone manufacturers, for example, shared 72% of the smartphone market among them in the first quarter of 2020 [36]. This high level of concentration in the market allows us to make a statement about the probability of LAP collisions.

With the probability of encountering a LAP collision $P(col)$, the probability of encountering a different OUI $P(dO)$ and the probability of encountering the same CID $P(sC)$, we first state that, since addresses are globally unique, encountering the same LAP twice means that the OUI is different. Therefore, for the probability of a LAP collision, we determine $P(col) = P(sC) * P(dO)$. Even without any further optimizations, this gives us a fairly small probability of around $P(col) \approx 5.96\text{e}{-8}$. Using our knowledge about the smartphone market, however, we can improve the odds even further. Let's assume that the 20 largest smartphone manufacturers share (almost) the entire market. In this case, $P(dO) \approx \frac{19}{20}$ and $P(col) \approx 5.66\text{e}{-8}$. Thus, if, in a certain environment, the system would register 10'000 different Bluetooth addresses, for example, the probability for a LAP collision would still only be $1 - (1 - P(col))^{10'000} \approx 0.06\%$. This is sufficient for the potential use cases of *BluePIL* and, therefore, the LAP as computed in [21] is used as a quasi-unique identifier for this work. This identifier is also suitable to identify individuals carrying a Bluetooth device since all devices in a piconet use the master device's LAP for the construction of the access code, *i.e.* two connected devices, such as a smartphone and a pair of Bluetooth headphones, do not produce two separate identifiers.

3.6 Device Localization

The following explains the device localization method used in the *BluePIL* system. A summary of the localization pipeline is presented in Figure 3.3. The methods used are described for four sensors, since this is the number that is assumed to be available (*cf.* Section 3.3), however, they are generally applicable to a larger amount of sensors as well.

3.6.1 Signal Strength Filter

As a first step in the device localization pipeline, the signal strength values obtained from the sensor are pre-processed. For this, it is important to understand the characteristics of RSS measurements. Figure 3.4 shows an example for RSS measurements obtained for a

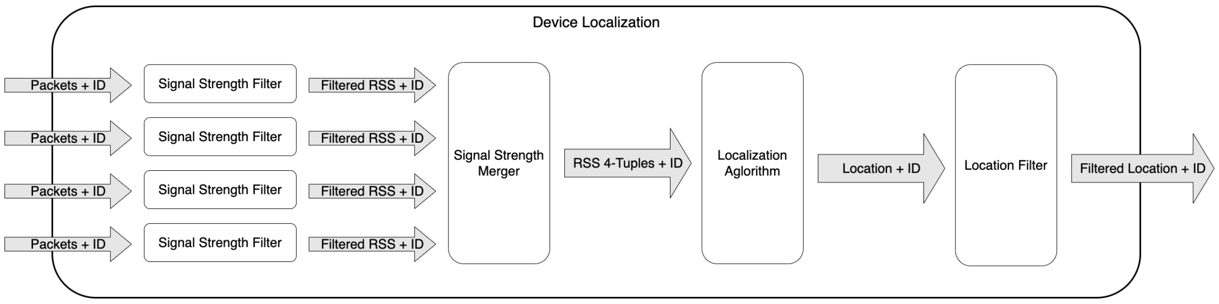


Figure 3.3: The topology of the device localization component

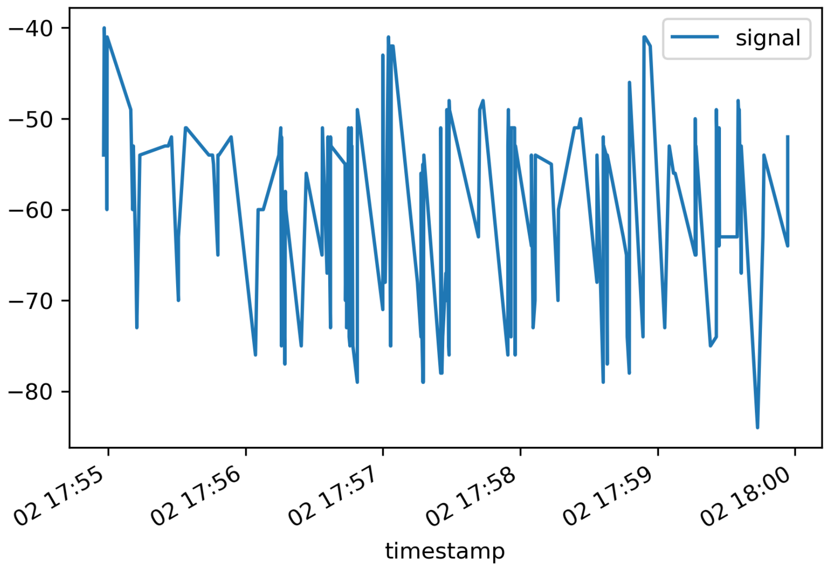


Figure 3.4: An example for RSS measurements obtained for a static device over a period of five minutes using an *Ubertooth* sensor

static device over a period of five minutes. This example illustrates the large amounts of high-frequency, high-variance noise that must be taken into account when working with this type of data. While part of the noise can be attributed to inaccuracies of the sensor, a significant amount of of disturbance originates from the effects of multipath fading, *i.e.* a signal may travel along multiple paths towards a sensor that diverge from the most direct path, the line of sight. This effect is caused, for example, by reflections of the signal on surfaces in the surroundings of the target device and the sensor. It makes the RSS a difficult value to work with since the distribution of this noise is not Gaussian, an assumption that many filtering and smoothing approaches work under. The main goal of this step in the processing pipeline is, therefore, the elimination of the noise caused by multipath fading and the conversion of the noise distribution to a Gaussian one.

Existing research suggests that the noisy parts of the RSS values correspond to the lower set of values in the RSS distribution (*cf.* Figure 3.5). In [25], a unidirectional outlier filter is found to be effective. It eliminates values that deviate from the maximum value by a certain degree. [22] determines the maximum to be the most effective filter for the pre-processing of RSS values for localization purposes. This makes sense on an intuitive level. The signal that travels along the line of sight, *i.e.* that is not influenced by

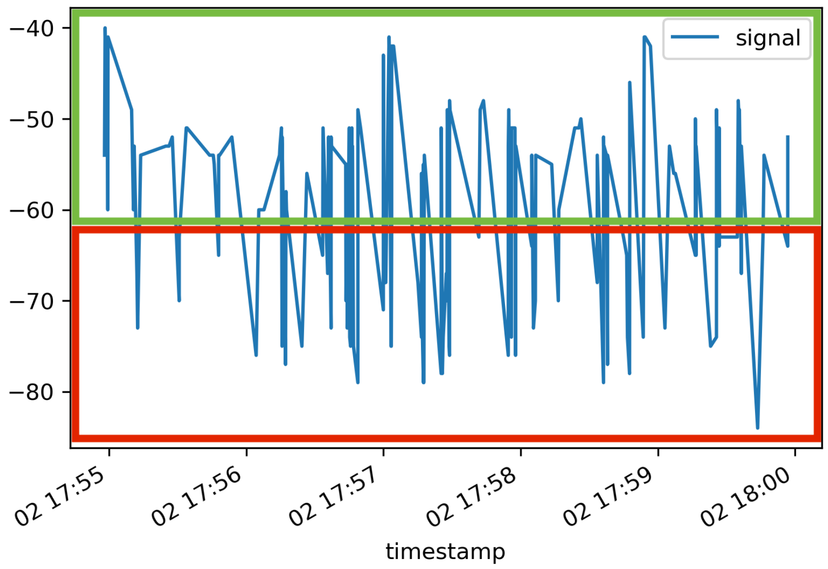


Figure 3.5: RSS values that are potentially useful (green) and those that are probably caused by multipath fading (red)

multipath fading, covers the smallest distance and thus arrives at the sensor with the highest strength. A combination of a maximum filter followed by a mean filter is, thus, used in *BluePIL*.

To account for the streaming paradigm described in Section 3.4, these filters work in a purely retrospective way, *i.e.* work with a local subset of the data that only uses values from the past. To this end, a rolling time window is implemented that only contains values from the interval $[t_c - \Delta t, t_c]$, with t_c the current time and Δt the window size, which is determined by the update frequency of the sensor, and the expected variance in the data.

3.6.2 Signal Strength Merger

To compute a location from the pre-processed signal strength values, a strategy has to be determined to merge the data streams for the four sensors used in this thesis. This part of the processing pipeline deals with two problems: First, the update cycles may differ between sensors, *i.e.* it cannot be assumed that all sensors will have the same amount of data available at a specific point in time. Second, the data delivered by the sensors may be fairly sparse. This may be due to the quality and capabilities of the sensors themselves, due to environmental factors or due to characteristics of the target device. The goal of this step is to deal with these problems, taking into account the streaming paradigm implemented for *BluePIL*.

Interpolation is able to help with both the problem of differing update cycles and sparsity of data. In general, we build upon the assumption that the update cycles of the individual sensors are short enough to legitimize the linear interpolation between two data points as a valid estimation of the true state of the system. To enable the inference of RSS values at a certain point in time through interpolation, measured values must be available preceding and succeeding said point. The signal strength merger will, therefore, delay the emission of a value from a sensor until data is available from all other sensors before and after the point in time where the value was received. Algorithm 3.1 shows the method used to achieve this and is explained briefly in the following.

The main while block (Lines 3-21) gets executed as long as data is available from the streams for the four sensors. We first check whether there has been a value registered for the sensor previously (Lines 5-6). If so, we check for all value sets that are waiting to be emitted whether they are missing a value for the sensor (Line 10). If they do, we interpolate a value linearly using the last and the current value-timestamp combination (Line 11). If there are no values missing from the value set, we emit it (Lines 12-14). We then store the current value for the next iteration (Line 16). Once at least one value has been registered for each sensor, we create a new value set to be stored as a waiting emission (Lines 17-20).

This algorithm ensures that the data, which is required to make inferences about the state of other sensors, is available at the point of emission of each individual sensor. This increases the accuracy and plausibility of our computed data points. It is superior to trivial strategies, for example using the last available value, in this regard. It does, however, disagree with the real-time requirements to some degree, changing *BluePIL* into what can be described as a near-real-time system. Considering that the delay introduced by the merger component will be small, as long as the sensors update frequently, this is a worthwhile trade-off.

3.6.3 Localization Algorithm

The evaluation of the related work in the topic of device localization (*cf.* Subsection 2.2.2) shows that all existing approaches rely on some sort of calibration or collaboration with the target device, *e.g.* to receive signals emitted by beacons or to be used in advance to calibrate the channel parameters. Since *BluePIL* needs to be completely passive to satisfy the goals defined for this thesis, this is impossible in our case. Our information is limited to the signal strengths that we can detect on an external sensor from any ongoing Bluetooth connection that we can passively eavesdrop on. This rules out any fingerprinting-based approaches, since they require the creation of a radio map with the devices involved beforehand, leaving the path-loss-model-based approaches. They too, however, suffer from a problem: The log-distance path loss model (*cf.* Subsection 2.1.7) requires parameters n and RSS_C to be defined beforehand in order to calculate a distance from a signal strength value. Based on existing research [27] [25], it is viable to set n to a fixed value based on the environment *BluePIL* is working in, as long as this does not change drastically. n is dependent on environmental factors and does not vary between devices. The issue with RSS_C, however, is not so easy to solve. Transaction strengths may vary between Bluetooth devices. Due to adaptive power control, they may even change over time for the same device [4]. The choice of a fixed value for RSS_C is, therefore, not an option.

Algorithm 3.1: The signal strength merger algorithm for four sensors

```
   // A set containing the values last received for each sensor
1  lastValues ← $\{null, null, null, null\}$;
   // A list of incomplete value sets waiting for emission
2  waitingEmissions ← ∅ ;
   // Repeat the following until the sensors are stopped
3  while A sensor has data to deliver do
4      value, timestamp, sensorIndex ← ReceiveData() ;
5      last ← lastValues[sensorIndex] ;
       // We cannot interpolate values unless there is a previous value
6      if last is not null then
7          lastValue, lastTimestamp ← last ;
           // Check all value sets that are still missing values
8          for waiting in waitingEmissions do
9              waitingValues, waitingTimestamp ← waiting ;
               // Interpolate a value if it is missing from the value set
10             if waitingValues[sensorIndex] is not null then
11                 waitingValues[sensorIndex] ←
                    Interpolate(lastValue, lastTimestamp, value, timestamp, waitingTimestamp)
                    ;
               // Emit the waiting emission if there are no values missing
12             if waitingValues does not contain null then
13                 Emit(waiting) ;
14                 waitingEmissions ← waitingEmissions − waiting ;
15         end
16     lastValues[sensorIndex] ← {value, timestamp} ;
       // Add new waiting emissions once we have data for all sensors
17     if lastValues does not contain null then
18         newValues ← $\{null, null, null, null\}$ ;
19         newValues[sensorIndex] ← value ;
20         waitingEmissions ← waitingEmissions + {timestamp, newValues} ;
21 end
```

To approach this, a method was designed that dynamically estimates the location of a Bluetooth device, and the necessary channel parameters of the path loss model. With k the number of sensors, (x_i, y_i) the location of the i-th sensor, (x, y) the location of the target device and d_i the distance between the i-th sensor and the target device, we can define the following multilateration problem (*cf.* Subsection 2.1.6):

$$(x - x_i)^2 + (y - y_i)^2 = d_i^2, i \in 1..k \tag{3.1}$$

Since we cannot compute d_i from the path loss model directly due to the issues mentioned before, we solve the path loss model equation for distance and then combine it with the multilateration problem above.

$$\begin{aligned} RSS(d) &= RSS_C - 10n\log(d) \\ d &= 10^{\frac{RSS_C - RSS(d)}{10n}} \end{aligned} \tag{3.2}$$

$$(x - x_i)^2 + (y - y_i)^2 = 10^{\frac{RSS_C - RSS(d_i)}{5n}}, i \in 1..k \tag{3.3}$$

We can now define a non-linear set of k minimizable equations in terms of x, y and RSS_C, with RSS_C the calibration signal strength 1 m away from the target device and RSS_i the RSS measurement for the i-th sensor.

$$f_i(x, y, RSS_C) = (x - x_i)^2 + (y - y_i)^2 - 10^{\frac{RSS_C - RSS_i}{5n}} \overset{!}{=} 0, i \in 1..k \tag{3.4}$$

This corresponds to a problem that can be solved using a non-linear optimization algorithm. *BluePIL* uses *Levenberg-Marquardt (LM)*, an iterative minimizer that can be described as a combination of the *Steepest Descent* and the *Gauss-Newton* methods [37] [38]. With $\mathbf{p} = (x, y, RSS_C)$, our parameter vector, we try to determine the vector $\mathbf{p}^+$ where $f_i(\mathbf{p}^+)$ is minimal for all i. LM works through a local linearization of the non-linear set of equations at a certain area of interest according to the statement $f(\mathbf{p} + \delta_p) \approx f(\mathbf{p}) + \mathbf{J}\delta_p$, where $\mathbf{J}$ is the *Jacobian* matrix. For our set of equations, the Jacobian matrix is defined as:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial RSS_C} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_i}{\partial x} & \frac{\partial f_i}{\partial y} & \frac{\partial f_i}{\partial RSS_C} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_k}{\partial x} & \frac{\partial f_k}{\partial y} & \frac{\partial f_k}{\partial RSS_C} \end{bmatrix} = \begin{bmatrix} 2(x - x_1) & 2(y - y_1) & -\frac{log10}{5n} * 10^{\frac{RSS_C - RSS_1}{5n}} \\ \vdots & \vdots & \vdots \\ 2(x - x_i) & 2(y - y_i) & -\frac{log10}{5n} * 10^{\frac{RSS_C - RSS_i}{5n}} \\ \vdots & \vdots & \vdots \\ 2(x - x_k) & 2(y - y_k) & -\frac{log10}{5n} * 10^{\frac{RSS_C - RSS_k}{5n}} \end{bmatrix} \tag{3.5}$$

With the Jacobian defined, LM then iteratively adjusts $\mathbf{p}$ by δ_p in a descending direction until convergence is reached. To ensure that this convergence is to a global minimum, an appropriate starting point $\mathbf{p}_0$ has to be defined. For the problem posed, it is important that the minimum is found in the area of overlap of all sensors. To guaratee this, $\mathbf{p}_0$ is chosen at the center of the area spanned by the sensors and with a value for RSS_C that approximates the range of values that we expect from the relevant device class.

$$\mathbf{p}_0 = (\frac{\sum_i x_i}{k}, \frac{\sum_i y_i}{k}, -30) \tag{3.6}$$

Due to the limited resources available for this thesis, the problem is generally limited to four sensors, *i.e.* $k = 4$. The localization algorithm may therefore also be referred to as a *quadlateration* algorithm in the following.

3.6.4 Location Filter

After having calculated a location in the previous step, we can now use our knowledge of the motion of a person carrying a Bluetooth device to improve these results further. Kalman filters are a popular method for the improvement of positioning calculations and have been used in many path-loss-based localization approaches [25] [24] [13]. They combine models for the state of the system, the knowledge of previous observations and models for the observation of states to estimate the most plausible state of a system that we can only capture through noisy observations (*cf.* Subsection 2.1.5).

To use a Kalman filter, we need to define the following: the state transition model F_k, the observation model H_k, the process noise covariance Q_k and the observation noise covariance R_k. We define our system's state using a simple kinematic model with (x_k, y_k) the current location's coordinates, and $(\dot{x_k}, \dot{y_k})$ the current velocity in x and y direction. The state vector, designed in a similar manner to [24], is then defined as follows:

$$s_k = \begin{bmatrix} x_k \\ \dot{x_k} \\ y_k \\ \dot{y_k} \end{bmatrix} \tag{3.7}$$

In accordance with Equation 2.4 and with Δt_k the time difference to the last state estimate s_{k-1}, we can then define the following state transition matrix:

$$F_k = \begin{bmatrix} 1 & \Delta t_k & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

This expresses a belief that the subject carrying a Bluetooth device will have moved in the direction gathered from the last measurement and that the velocity of said movement will not have changed abruptly. For our process noise covariance, we will use discrete white noise as is suggested in [11] and [39] under the assumption that the noise is a

Wiener process, *i.e.* is independent from previous time intervals and constant over a time interval. With the variance $\sigma_v^2 = 0.001$ [11], it is defined as follows:

$$Q_k = \begin{bmatrix} \frac{1}{4}\Delta t_k^4 & \frac{1}{2}\Delta t_k^3 & 0 & 0 \\ \frac{1}{2}\Delta t_k^3 & t_k^2 & 0 & 0 \\ 0 & 0 & \frac{1}{4}\Delta t_k^4 & \frac{1}{2}\Delta t_k^3 \\ 0 & 0 & \frac{1}{2}\Delta t_k^3 & t_k^2 \end{bmatrix} * \sigma_v^2 \tag{3.9}$$

We will use the values obtained from the previous step in the pipeline as our observations, *i.e.* the location estimates calculated through our modified multilateration method. We, therefore, use the following observation vector:

$$z_k = \begin{bmatrix} x_k \\ y_k \end{bmatrix} \tag{3.10}$$

In accordance with Equation 2.5, we then define the following observation matrix to express that the observation corresponds to the x and y coordinates of our state vector:

$$H_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.11}$$

Finally, we design the observation noise covariance matrix. The values used were determined experimentally and work well with sensors used for this thesis. Using a different set of sensors, they might have to be adjusted.

$$R_k = \begin{bmatrix} 0.3 & 0 \\ 0 & 0.3 \end{bmatrix} \tag{3.12}$$

Using the Kalman filter described allows us to improve the values calculated in the previous step using the information contained in previous values and our knowledge of the system dynamics. It eliminates outliers and smooths the results simultaneously, using plausibility as a determining factor.

Chapter 4

Implementation

4.1 Hardware

Four Ubertooth One devices (*cf.* Subsection 2.1.4) are used as sensors for the *BluePIL* system. As per the assumptions made for *BluePIL* (*cf.* Section 3.3), it allows access to the lower layers of the BTBR/EDR protocol at a comparatively low cost. This enables the device identification via the LAP (*cf.* Section 3.5) through the AC, which is normally hidden behind a layer of abstraction when using a conventional Bluetooth module. It is not limited to the passive monitoring of the advertisement channels, and allows the manual selection of any of the the 79 BTBR/EDR channels. Packets can then be captured from said channel, are decoded on the sensor and then transferred to the host system via USB along with some meta-information, such as RSS values. *Project Ubertooth* includes both firmware for the sensor as well as C applications and libraries to run on the host system as a counterpart.

Four *Asus Tinkerboards* [40] were used to take up the role of the nodes (*cf.* Section 3.4). The Tinkerboard includes a *Rockchip RK3288* System on Chip (SoC) with a Quad-Core 1.8 GHz processor, 2 GB of dual-channel DDR3 memory in a standard single board computer (SBC) form factor which provides plenty of processing power for the *BluePIL* use case. USB 2.0 ports allow it to interface with the Ubertooth sensors and an integrated 802.11 b/g/n Wi-Fi chip enables the interconnection with the sink requiring minimal cabling work. The Tinkerboards were configured with *armbian* [41], a low-profile *Debian Linux* version optimized for ARM development boards.

For reasons of convenience and availability, a 2017 *MacBook Pro* [42] was used as the sink (*cf.* Section 3.4). With a 3.3 GHz dual-core *Intel i5* processor and 16 GB of DDR3 memory, its processing resources are more than sufficient for its role in the *BluePIL* system. Other lower-performance and smaller-form-factor options, such as the Asus Tinkerboard, could be valid replacements.

The final component in the *BluePIL* hardware setup was a *GL-iNet MiFi Smart Router*, a low-profile, portable WiFi access point and router. It was used to create a wireless network for the nodes and the sink to communicate over. USB tethering with an Android smartphone provided internet access for time-synchronization and deployment of the application.

4.2 Software

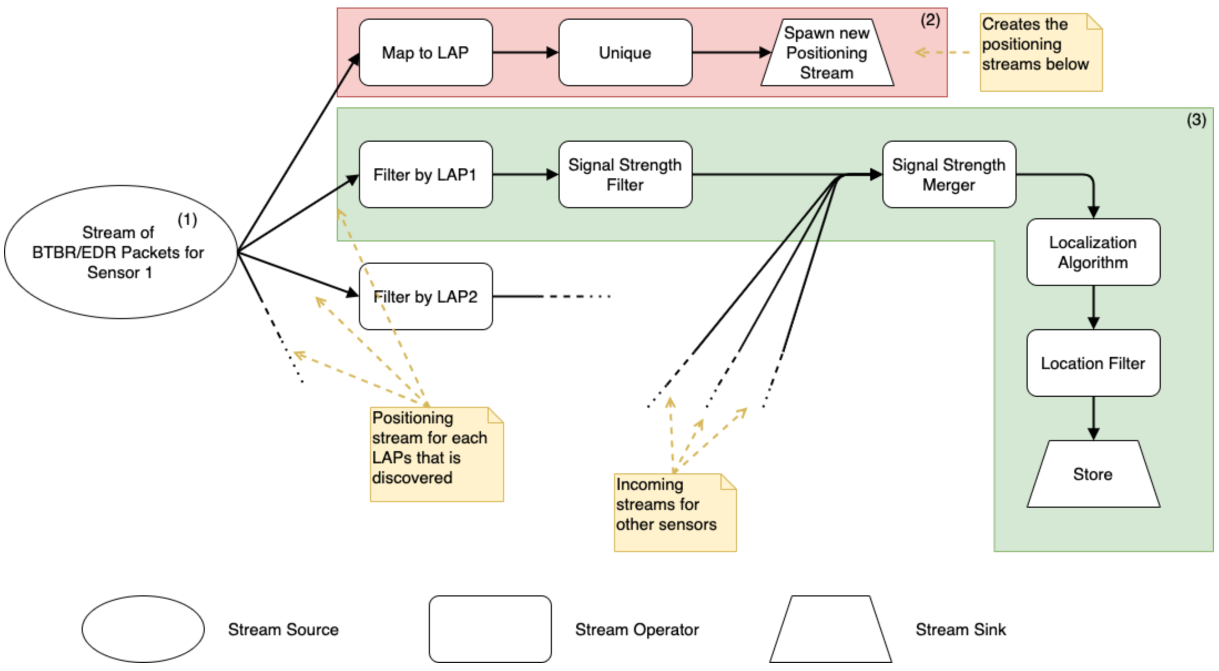
The *BluePIL* system described in Chapter 3 was implemented as an asynchronous, distributed *Python 3* application. Python brings along with it built-in facilities for concurrency and asynchronous processing using the *asyncio* library as well as a rich ecosystem of libraries for numerical mathematics and data analysis. The fact that it is an interpreted language also simplifies the deployment over multiple different architectures since the code does not have to be re-compiled with every change. The application consists of two main modules, the node component and the sink component, running on separate devices, that are interconnected through an internet protocol (IP) network. The system is designed, however, in a way that allows the application to be run on a single device as well, should there be need. The sink and the nodes are set up in a master-slave architecture, where all configuration is performed via the master, who in turn configures the nodes autonomously. This ensures that the configuration and initialization of the application are as easy as possible. As described in Section 3.4, the streaming paradigm was implemented throughout the application to enable the real-time evaluation of incoming data from the sensors.

4.2.1 Data Streams

The streaming architecture described in Section 3.4 was implemented using *Streamz*, a Python framework for the management of continuous streams of data that includes operators for transformations, branching, joining, flow control, etc. and can be used in an asynchronous manner[43]. Streamz is extensible through the definition of custom operators, where each operator receives items from one or more upstream operators, performs some operation on them and emits zero or more items downstream. Operators are implemented as Python classes and may, therefore, include state. The *BluePIL* processing pipeline was then implemented as a combination of built-in and custom Streamz operators.

Figure 4.1 shows a slightly simplified version of the data streams implemented in the *BluePIL* application and will be explained here. The output of the sensor is emitted from the stream source (1). A set of operators is then set up that extracts the LAP values from the packets in the stream, such that each unique value is only emitted once (2). For each of the unique LAP values emitted by the stream, a new set of operators is connected to the stream source (3). This set of operators will perform the location computation according to Section 3.6, first filtering the signal strength values, then merging them with the values from other sensors, computing a location and finally filtering said location. As a last step, the results of the computation can be stored.

In practice, the setup is sightly more complicated since the streams pass through multiple devices, *i.e.* the nodes and the sink. The streaming architecture, however, is still valid



Map to LAP
Unique
Spawn new Positioning Stream
(2)
Creates the positioning streams below
(3)
Filter by LAP1
Signal Strength Filter
Signal Strength Merger
Localization Algorithm
Location Filter
Store
Stream of BTBR/EDR Packets for Sensor 1
(1)
Filter by LAP2
Positioning stream for each LAPs that is discovered
Incoming streams for other sensors
Stream Source
Stream Operator
Stream Sink

Figure 4.1: A slightly simplified representation of the data streams used in the *BluePIL* implementation

overall. Theoretically, the data could pass through the network at any connection between two stream operators without impeding the functionality of the architecture. The concrete implementation of the connection between nodes and sink is explained in Subsection 4.2.3.

4.2.2 Interface with the Sensor

The Ubertooth sensor is connected to the host system via USB. Project Ubertooth includes a host application and libraries written in C, that interface with the Ubertooth, send control signals to it and receive captured packets. While this was useful in the initial experimentation with the sensor, a Python implementation was needed to profit from the advantages offered by this language (*cf.* Section 4.2). Some work porting the Ubertooth host code to Python has already been done and is available publicly [44]. The implementation is, however, incomplete, outdated, and in some cases faulty. It was, therefore, used as a starting point, corrected and added to for *BluePIL* by reverse-engineering the Ubertooth host C libraries. The main areas that needed improvement were the following:

De-Serialization: The publicly-available implementation only offered incomplete de-serialization of the packets from the sensor, *i.e.* some values in the binary data received via USB were not made available. This was amended by matching the interpretation of the binary data to the format known from the Ubertooth firmware code, where the corresponding USB packet is defined as a C struct (*cf.* Listing 4.1). This was accomplished using the Python *struct* library which only requires the definition of the data types used in the C struct and the byte order.

RSS Values: The RSS values received from the Ubertooth in a USB packet (*cf.* Listing 4.1: `rssi_max`, `rssi_min`, `rssi_avg`) tend to be unusable without further processing.

Listing 4.1: USB packet format defined in the Ubertooth firmware [7]

```
typedef struct {
        uint8_t   pkt_type;
        uint8_t   status;
        uint8_t   channel;
        uint8_t   clkn_high;
        uint32_t  clk100ns;
        int8_t    rssi_max;    // Max RSSI seen while collecting
                               // symbols in this packet
        int8_t    rssi_min;    // Min ...
        int8_t    rssi_avg;    // Average ...
        uint8_t   rssi_count;  // Number of ...
        uint8_t   reserved[2];
        uint8_t   data[DMA_SIZE];
} usb_pkt_rx;
```

While the Ubertooth can sense an RSS value, it has no way of directly linking it to a packet. Therefore, it merely aggregates the RSS values measured over the time it was collecting symbols for a packet. This, however, leaves a certain amount of inaccuracy behind, creating RSS values that do not correspond to the actual environment. The Ubertooth host code, therefore, includes an additional aggregation step, choosing the maximum `rssi_max` over 10 detected packets as the true value. The RSS values are also unitless when they are received from the sensor and have to be converted to a value in decibel-milliwatts (dBm) using a piecewise linearization of a curve defined for the CC2400 wireless transceiver.

Asynchronous Version: For the Python implementation of the Ubertooth host code to be able to deal with multiple sensors simultaneously, an asynchronous version of the data stream from the sensor had to be created. So far, it had been implemented as a Python generator expression. The conversion involved upgrading the source from Python 2 to Python 3. An asynchronous generator expression could then be used to accomplish the task at hand.

Clock Drift: During experimentation with the Ubertooth sensor, a few problems with the transmitted clock values were discovered. Apart from frequent overflows of the corresponding field in the USB packet (*cf.* Listing 4.1: `clk100ns`), a noticeable drift of the values compared to the device clock on the host system was apparent. The decision was, therefore, made to include the host system clock at the time when the USB packet was received as a timestamp in the sensor data stream.

In addition to the changes made to the existing implementation, a strategy for channel selection had to be defined. To account for the possibility that certain channels may be avoided during an active connection due to Bluetooth's adaptive frequency hopping, the Ubertooth sensors are configured to switch between channels randomly during operation.

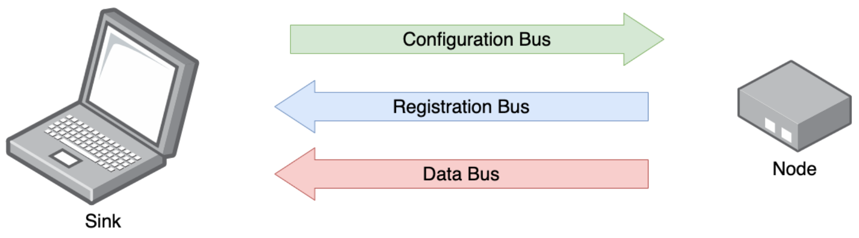


Figure 4.2: The available channels between the node and the sink

4.2.3 Node-Sink Communication

The communication between the nodes and the sync was implemented using *Python asyncio tcp streams* [45]. This allowed it to be quick to implement, asynchronous and performant. Three unidirectional channels were created for each node-sink combination, as is illustrated in Figure 4.2: The configuration bus going from sink to node and the registration and the data bus going from node to sink. This was accomplished by opening a TCP port on every node to receive configuration messages and two TCP ports for every node on the sink to receive registration and data messages. The configuration bus is used for the implementation of the master-slave architecture (*cf.* Section 4.2). Any information needed by the node to connect the registration and data buses to the sink, and signals to start or stop sending data are dispatched via this channel. The registration bus is used to send any notice of newly detected LAPs to the sink. This information is needed to set up the corresponding endpoints for the data stream. The data bus is used to send the data stream, *i.e.* the filtered signal strength values for any LAPs detected.

A sequence diagram of the interactions between sink, node and sensor is shown in Figure 4.3. The system is initialized by the sink sending a configuration message containing the start signal and information on the connection parameters to the node. The node in turn messages the sensor, which then starts emitting captured packets. As long as the system is not stopped, the following is then repeated: The node receives captured packets from the sensor. It performs some first pre-processing on the packet. If the LAP encountered is not known, the node sends a registration message to the node containing the newly discovered LAP. This enables the sink to prepare the incoming data stream for dealing with said LAP value. The packet is then encapsulated into a data message and sent to the sink, who processes it further. Once the system is stopped, the sink sends another configuration message to the node, which contains the stop signal. The node then stops the sensor and the system is terminated.

Finally, Figure 4.4 shows the connections between node and sink in the context of the streams introduced in Section 4.2.1. Both the stream of unique LAPs and the positioning streams have to be transmitted intermittently over the registration bus and the data bus respectively. The stream of unique LAPs is branched, such that it can be used to spawn new positioning streams on the node and the sink. The positioning streams for individual LAPs are merged for the transmission over the data bus and split again after. This avoids having to create an excessive amount of channels between the node and the sink in order to maintain scalability when compared to spawning a new channel for each LAP detected. The merging and splitting are, however, transparent to the system overall, and the general streaming architecture described in Section 3.4 remains valid.

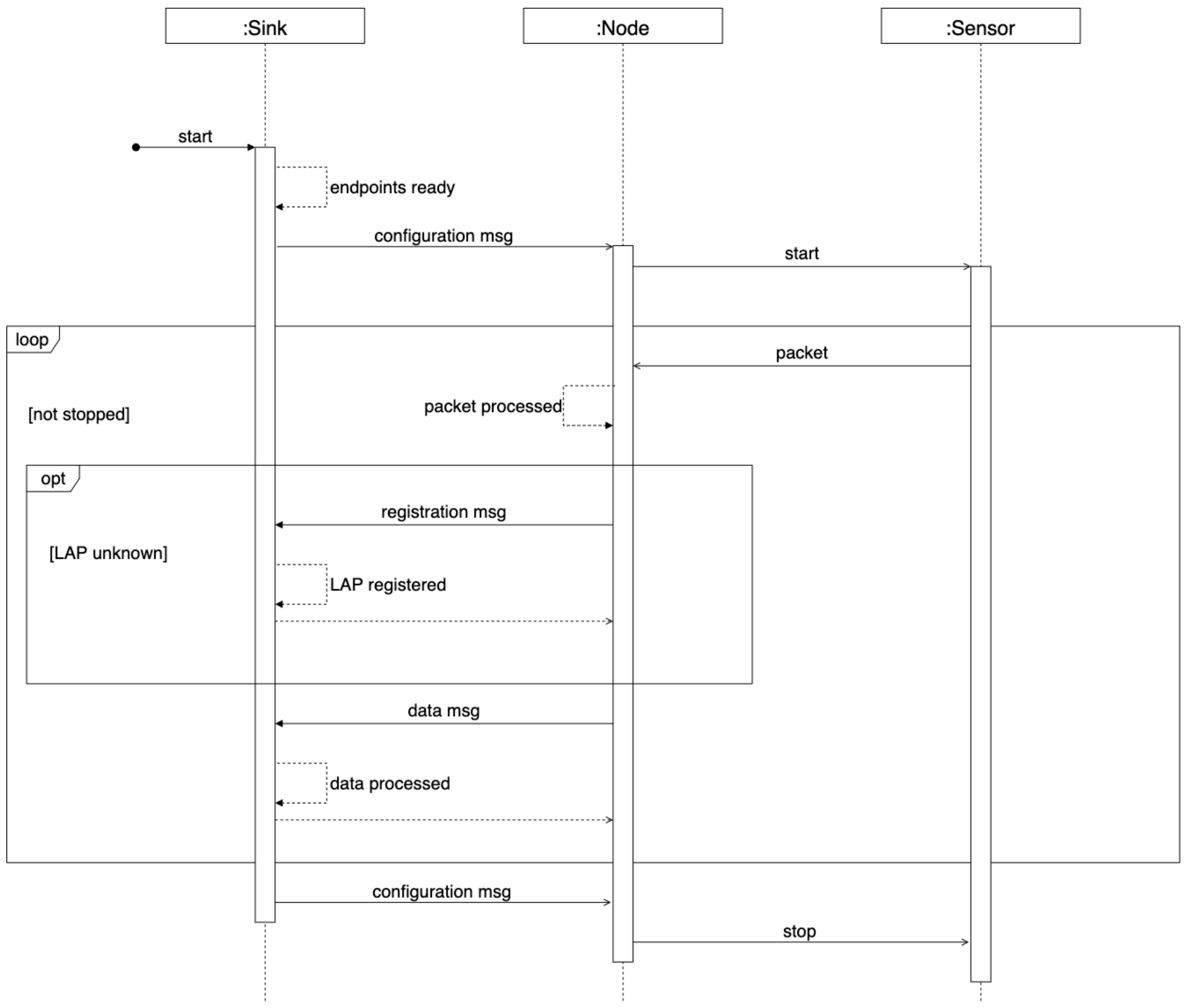


Figure 4.3: Sequence diagram summarizing the communication between the sink, a node and a sensor

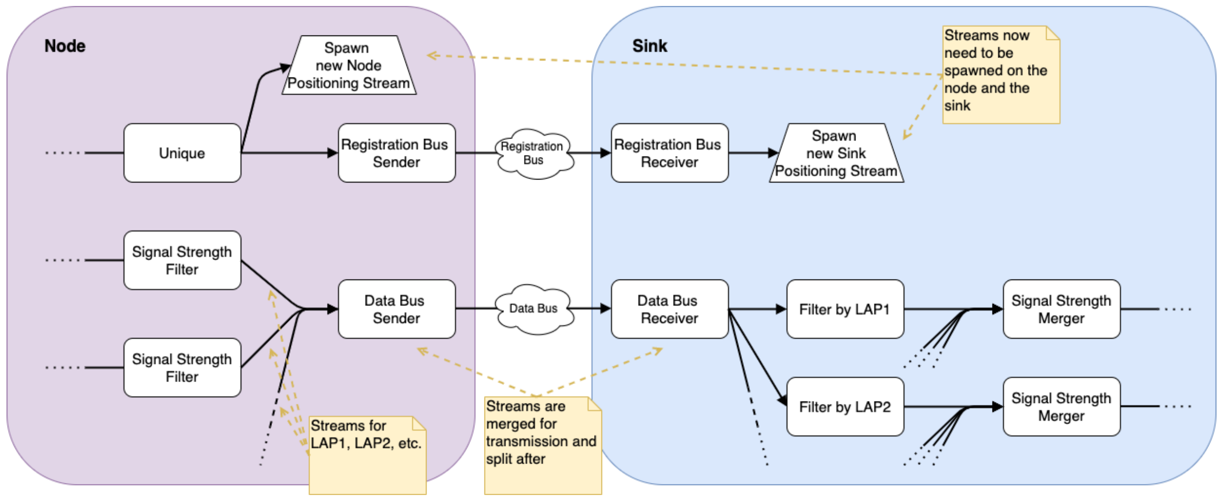

Node
Spawn new Node Positioning Stream
Unique
Registration Bus Sender
Registration Bus
Signal Strength Filter
Signal Strength Filter
Data Bus Sender
Data Bus
Streams for LAP1, LAP2, etc.
Streams are merged for transmission and split after
Sink
Streams now need to be spawned on the node and the sink
Registration Bus Receiver
Spawn new Sink Positioning Stream
Data Bus Receiver
Filter by LAP1
Filter by LAP2
Signal Strength Merger
Signal Strength Merger

Figure 4.4: Excerpt of Figure 4.1 showing where streams are interrupted and transmitted between node and sink

4.2.4 Device Identification and Localization

The approach described in [21] and selected for this thesis for device identification has been implemented in libbtbb [46], a Bluetooth baseband decoding Library, as part of Project Ubertooth. Similar to the interface with the Ubertooth (*cf.* Subsection 4.2.2), a partial port of said C library to Python has been implemented [44] and was consequently deployed in *BluePIL*. Only few modifications to this initial work were necessary. Specifically, the code contained an error where the entire USB packet was searched for the AC, whereas it was only necessary to search the Bluetooth packet content contained in the USB packet (*cf.* Listing 4.1: `data`). This caused no faults in the data but produced inefficiencies that are significant, given that a large number of potential packets has to be processed on the host system.

It was important to ensure that the device localization pipeline was as efficient as possible, given that this is computationally the most complex part of the *BluePIL* system. The implementation, therefore, relies heavily on *scipy* and *numpy* [47], two associated libraries that include optimized and tested versions of a wide range of numerical routines used in scientific computing. They manage to be fast thanks to the usage of non-interpreted languages, such as C, C++ or Fortran, for time-critical parts of the algorithms included and are, thus, used for the computation of the Levenberg-Marquardt algorithm (*cf.* Subsection 3.6.3). The *filterpy* Library [48] is used for the iterative evaluation of the Kalman filter. It also relies on numpy and scipy in the background and, therefore, brings along the same advantages. The interval used for the sliding window operations (*cf.* Subsection 3.6.1) was set to 20s, a value appropriate to the update frequency of the Ubertooth in the given environment.

Chapter 5

Evaluation

5.1 Experiment 1: Evaluation of the Device Localization Method

5.1.1 Experimental Setup

A first experiment was conducted to evaluate the effectiveness of the device localization method designed for this thesis (*cf.* Section 3.6). The experiment was performed both in an indoor and an outdoor space to compare the influence of these two environments. The scenes are shown in Figure 5.1. The indoor experiment was performed in a room in the author's apartment, that was at the time empty. This was ideal since it allowed to keep the amount of signal interference as low as possible and other indoor spaces were unavailable due to the outbreak of a global pandemic. The outdoor experiment was done on a private terrace in a residential area of Zürich. This allowed to decrease the amount of signal interference from multipath fading since more space was available. A $4.2\,\mathrm{m} \times 2.9\,\mathrm{m}$ area was designated to perform the experiments in. An Uberooth One sensor was placed at each of the four corners of this space and connected to a MacBook Pro via a 2.5 m USB cable. Nine points were then defined where static measurements would be carried out. This setup is summarized in Figure 5.2. The measurement points were chosen mostly on one side of the space. This was deemed sufficient due to the symmetry of the setup. A distance of at least one meter was maintained to the sensors for all measurements.

The experiment was then conducted as follows: A *Nokia 7 Plus* smartphone was connected to a pair of *JBL Reflect Flow* Bluetooth headphones. Music was streamed over said connection throughout the experiment to generate traffic that could be captured passively. In order to keep the conditions as realistic as possible, the headphones were placed in a test subject's ears and the smartphone in their front right pant pocket. The test subject then stood for 5 min at each of the nine points shown in Figure 5.2. The four Ubertooth One sensors were configured to record any packets that could be intercepted during that time interval.

(a) Outdoor Environment

(b) Indoor Environment

Figure 5.1: The spaces chosen for the first experiment

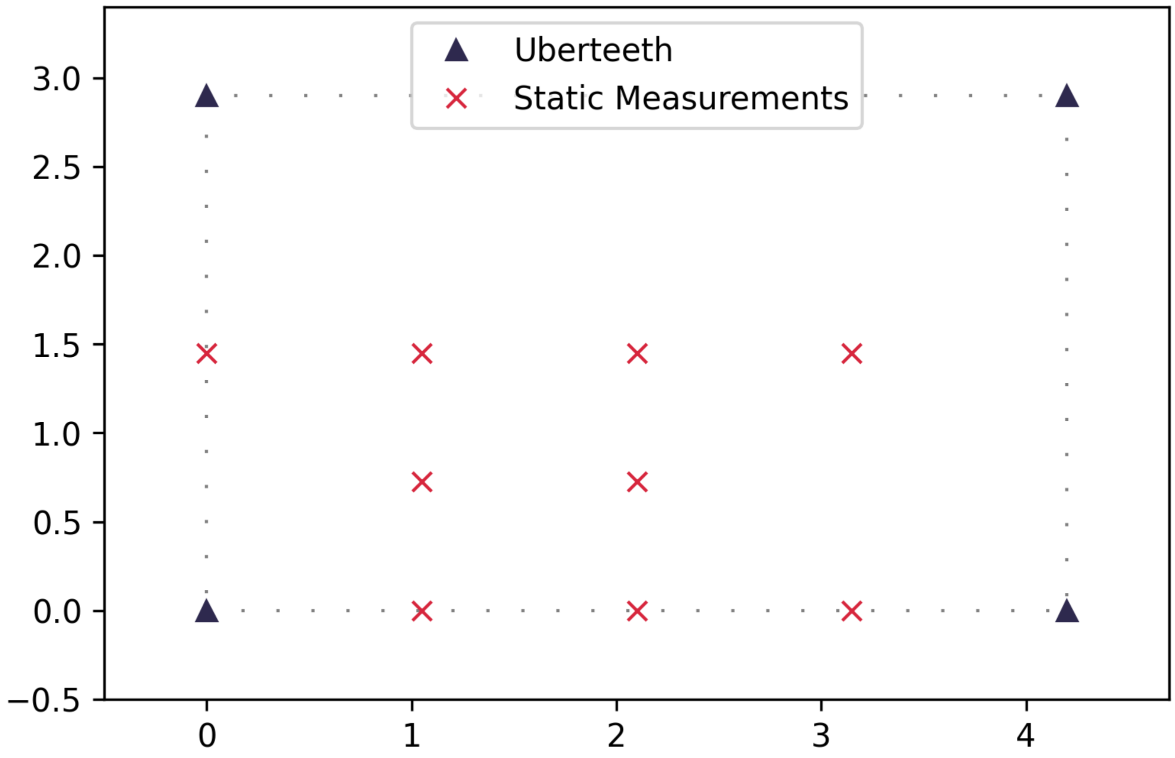


Figure 5.2: The setup for Experiment 1

True Point (m)	**Mean Estimated Point** (m, rounded)	**Mean Error** (m, rounded)	**Mean No. Measurements/Sensor**
$(2.10, 0.00)$	$(2.121, 0.254)$	0.398	439.5
$(1.05, 0.00)$	$(1.705, 1.538)$	1.703	354.75
$(3.15, 0.00)$	$(2.204, 0.591)$	1.150	397.5
$(2.10, 1.45)$	$(1.635, 0.480)$	1.153	334.0
$(1.05, 1.45)$	$(1.059, 0.716)$	0.855	306.5
$(3.15, 1.45)$	$(2.479, 0.423)$	1.302	284.0
$(0.00, 1.45)$	$(1.090, 0.983)$	1.195	132.75
$(1.05, 0.73)$	$(1.118, 1.294)$	0.835	285.75
$(2.10, 0.73)$	$(1.645, 0.840)$	0.769	244.75
Overall Mean Error: 1.040		**Overall Mean No. Meas./Sensor**: 308.83	

Table 5.1: Results from the outdoor environment in Experiment 1

This experiment used an initial static version of the localization method described in Section 3.6, *i.e.* the analysis of the data was done after the fact, not using the streaming system described in Section 3.4. This allowed for simplifications regarding the merging of the data sets from individual sensors. The streaming interpolation method used in the final processing pipeline could be omitted and the data could be merged using a static interpolation and re-sampling process. While this initial approach differs slightly from the final system, it does not invalidate the results of this experiment as an evaluation of the device localization method.

This experiment also included an evaluation of the filtering methods used in the BluePIL processing pipeline, namely the method used for signal strength filtering (*cf.* Subsection 3.6.1) and location filtering (*cf.* Subsection 3.6.4). The following variants were included for signal strength filtering: a simple rolling mean filter, a rolling maximum followed by a rolling mean filter, and a rolling maximum followed by a rolling median filter. With regards to location filtering, the improvement gained by the Kalman filter was analyzed.

5.1.2 Results

Tables 5.1 and 5.2 show the results of the outdoor and indoor experiments. The average location estimation, the average localization errors, and the average number of measurements per sensor for the Nokia smartphone's LAP are shown. Data for the point $(2.1, 1.45)$ is missing in the results from the indoor experiments. This is due to the failure of one of the sensors that was only noticed after the completion of the experiment.

Overall, the positioning accuracy in both experiments was fairly similar, with an average error of 1.04 m and 1.061 m for the outdoor and the indoor experiments respectively. The error values ranged from 0.398 m to 1.703 m for the outdoor and 0.612 m to 1.856 m for the indoor experiment. While the overall sensor performance was quite similar in the indoor and the outdoor experiment, collecting around one measurement per second, it was more stable in the indoor environment where the mean number of measurements per sensor for

True Point (m)	**Mean Estimated Point** (m, rounded)	**Mean Error** (m, rounded)	**Mean No. Measurements/Sensor**
$(2.10, 0.00)$	$(1.798, 0.826)$	0.888	258.5
$(1.05, 0.00)$	$(1.695, 1.692)$	1.856	253.25
$(3.15, 0.00)$	$(3.193, 1.629)$	1.718	206.25
$(2.10, 1.45)$	–	–	–
$(1.05, 1.45)$	$(0.942, 1.522)$	0.612	231.0
$(3.15, 1.45)$	$(2.761, 1.965)$	0.671	299.5
$(0.00, 1.45)$	$(0.556, 1.238)$	0.682	249.75
$(1.05, 0.73)$	$(1.118, 1.412)$	0.822	190.75
$(2.10, 0.73)$	$(1.931, 1.809)$	1.239	228.0
Overall Mean Error: 1.061		**Overall Mean No. Meas./Sensor**: 293.63	

Table 5.2: Results from the indoor environment in Experiment 1

Signal Strength Filtering	**Location Filtering**	**Mean Error** (m, rounded)
Mean	Kalman	1.043
Max + Mean	Kalman	1.040
Max + Median	Kalman	1.094
Max + Mean	–	1.050

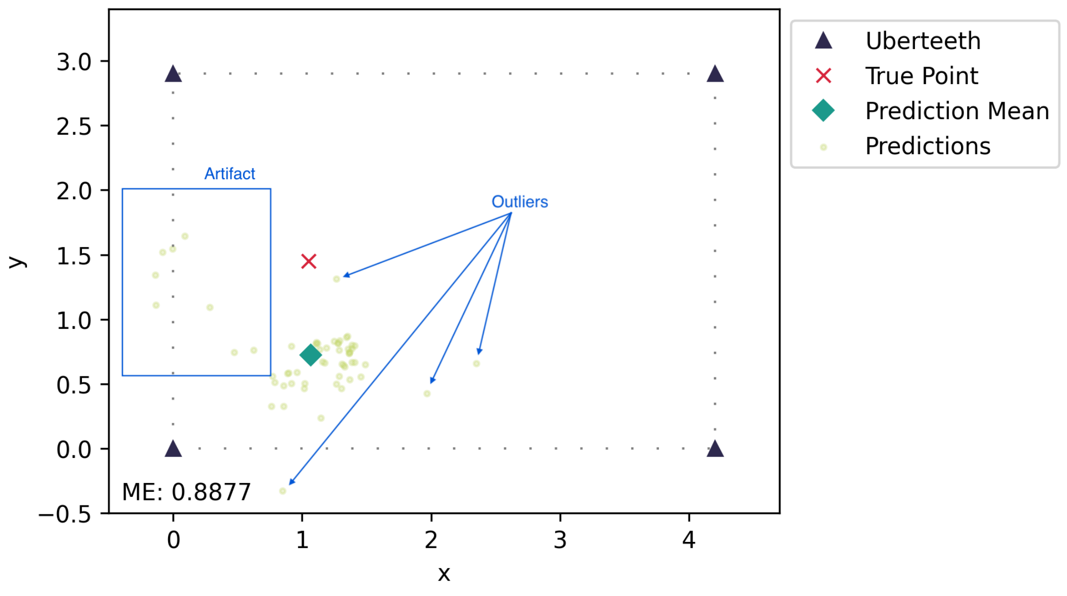
Table 5.3: Comparison of filtering methods in the outdoor environment in Experiment 1

the Nokia smartphone's LAP ranged from 190.75 to 299.5 compared to 132.75 to 439.5 in the outdoor case. Both sets of results show some outliers in the upper range of the error values, most notably point $(1.05, 0.00)$ for the outdoor experiment and points $(1.05, 0.00)$ and $(3.15, 0.00)$ for the indoor experiment.

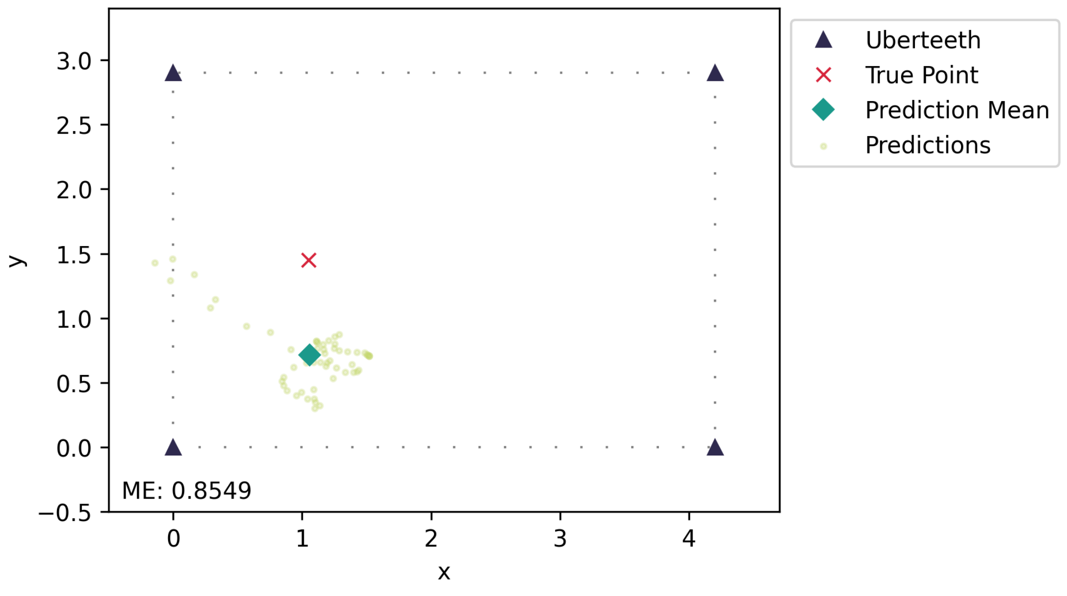
Tables 5.3 and 5.4 show the comparison of filtering methods for the outdoor and indoor experiments respectively. The analysis of the signal strength filtering methods confirms the findings mentioned in [22], establishing the maximum as the most effective filtering method, albeit by a small margin. Results show that the combination of a rolling maximum followed by a rolling mean outperforms other signal strength filtering approaches tested, *i.e.* a rolling mean filter and a maximum-median combination. This supports the hypothesis described in Subsection 3.6.1 regarding the effects of multipath fading on RSS values. Results also support the effectiveness of the Kalman filter in the location filtering step. It is shown to improve the overall positioning accuracy in both the indoor and the outdoor experiments. While this improvement is relatively small concerning the overall mean localization error, a visual inspection of the location estimates shows the Kalman filter's true value. Figure 5.3 contains an example from the outdoor experiment, which will be used to illustrate this. The localization results without Kalman filtering (*cf.* Subfigure 5.3a) exhibit a number of implausible outliers and an artifact where a series of points diverges from the prediction mean. The version with Kalman filtering (*cf.* Subfigure 5.3b) removed the outliers and converted the predictions in the artifact to a path that is more probable to correspond to the movement of a person carrying a Bluetooth device, *i.e.* a coherent motion in a certain direction.

Signal Strength Filtering	**Location Filtering**	**Mean Error** (m, rounded)
Mean	Kalman	1.075
Max + Mean	Kalman	1.061
Max + Median	Kalman	1.082
Max + Mean	–	1.067

Table 5.4: Comparison of filtering methods in the indoor environment in Experiment 1



(a) Localization without Kalman filtering



(b) Localization with Kalman filtering

Figure 5.3: An illustrative example for the differences between localization with and without Kalman filtering from Experiment 1

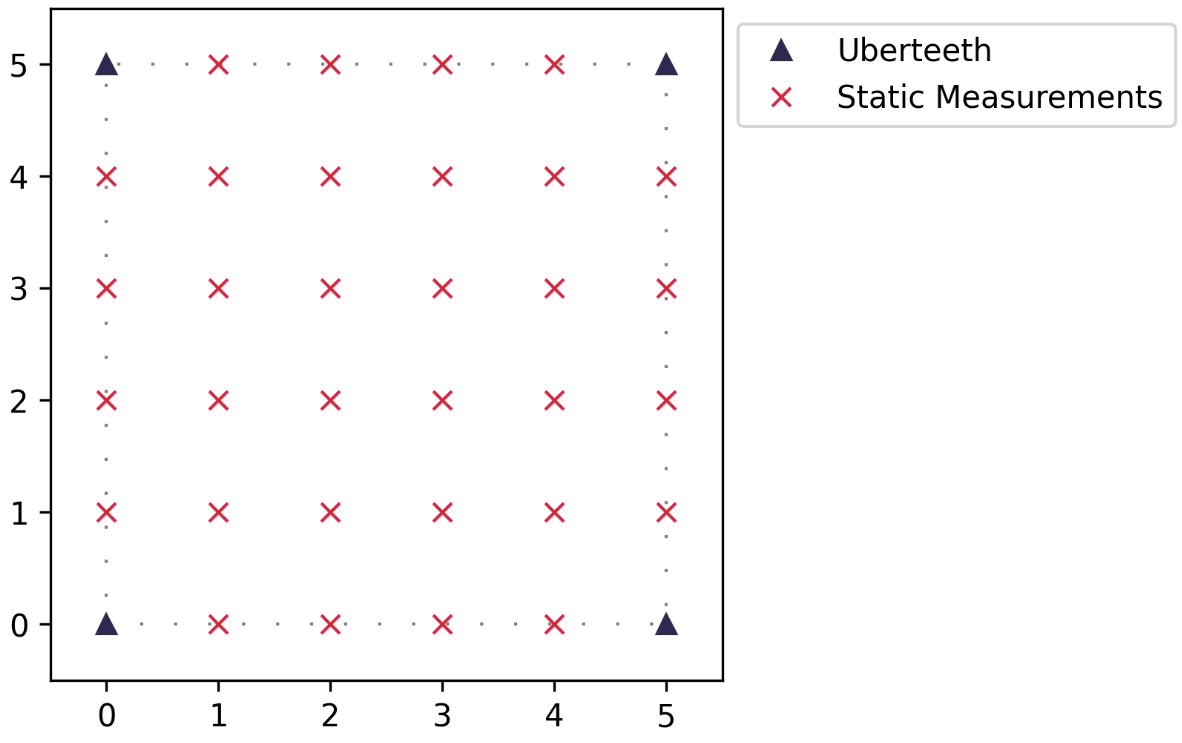


Figure 5.4: The setup for static measurements in Experiment 2, Part 1

5.2 Experiment 2: More Challenging Conditions & Evaluation of the System Design

5.2.1 Experimental Setup

A second experiment was designed to evaluate the *BluePIL* device localization method under more challenging conditions than the first experiment and to evaluate the system's performance in its final streaming architecture. A 5 m × 5 m space was designated to perform the experiment in, located in the same outdoor environment as the corresponding part of Experiment 1 (*cf.* Figure 5.1a). An Ubertooth One sensor was placed at each corner of the space, connected to an Asus Tinkerboard. A MacBook Pro was used to control these nodes, as is described in Chapter 4. The experiment was then performed in two parts:

A first step aimed to repeat Experiment 1 under more challenging conditions. 32 equally spaced points were chosen in the 5 m × 5 m space. They are shown in Figure 5.4. As in Experiment 1, a test subject holding a Nokia smartphone, which was streaming audio to a pair of JBL Bluetooth headphones, traversed these points, resting at each one for one minute. During this minute, data was captured by the Ubertooth sensors. This data was then analyzed ex-post with the same static version of the pipeline used in Experiment 1.

The goal of the second step was to test the system in its full streaming implementation, as described in Section 3.4. To this end, five points were chosen in the 5 m × 5 m space, as

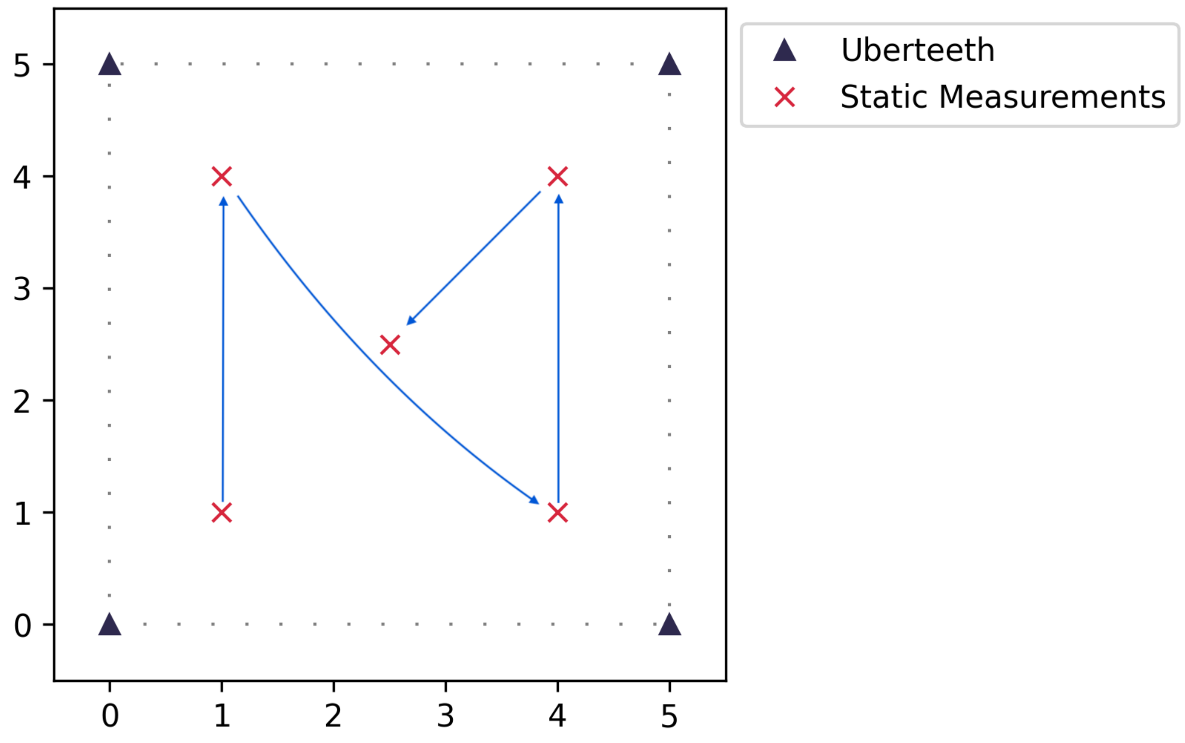


Figure 5.5: The setup for static measurements in Experiment 2, Part 2 and the movement between points

is shown in Figure 5.5. Again, the Nokia smartphone and the JBL headphones were used to stream audio over Bluetooth. The near-real-time positioning pipeline was then run for 15 minutes. During this time, the test subject covered each of the five points, resting at each point for two minutes and taking a maximum of one minute for the change between points. One minute of buffering time was included at the beginning. The Points were traversed according to the following order: $(1, 1) \rightarrow (1, 4) \rightarrow (4, 1) \rightarrow (4, 4) \rightarrow (2.5, 2.5)$.

5.2.2 Results

Experiment 2 revealed concerns with the Ubertooth sensors used. Most importantly, the performance regarding number of packets captured deteriorated significantly from Experiment 1. During the first part of the experiment, the number of packets captured per second and per sensor was reduced to about 0.38 compared to 1.0 from before. This created issues in the location computation, especially in the first part of the experiment: Due to the decreased number and the fact that the captured packets were not evenly spread throughout the time interval, it occurred that, for some of the points, there was no overlap between the points in time of the RSS measurements. Consequently, it was impossible to merge the RSS value streams between sensors. Only points that produced a sufficient overlap of a minimum of ten seconds were, therefore, analyzed for the first part of the experiment.

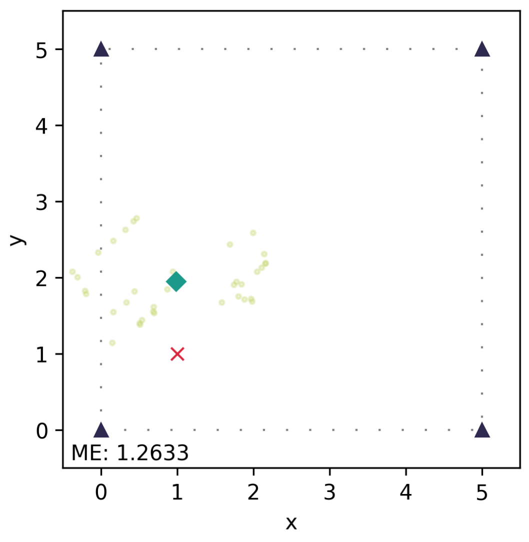
Table 5.5 shows the results for the first part of the experiment. It displays an obvious

True Point (m)	**Mean Estimated Point** (m, rounded)	**Mean Error** (m, rounded)	**Mean No. Measurements/Sensor**
$(0, 2)$	$(2.203, 2.426)$	2.244	20.25
$(1, 0)$	$(1.438, 1.681)$	1.737	35.0
$(1, 2)$	$(1.720, 1.888)$	0.729	36.0
$(2, 4)$	$(3.232, 4.619)$	1.379	30.0
$(3, 1)$	$(1.243, 1.738)$	1.905	17.25
$(4, 1)$	$(4.429, 2.255)$	1.326	22.0
$(4, 2)$	$(3.193, 1.876)$	0.816	19.25
$(4, 3)$	$(4.378, 3.073)$	0.385	21.5
$(4, 4)$	$(4.806, 2.629)$	1.590	26.75
$(5, 4)$	$(2.292, 2.787)$	2.967	25.0
Overall Mean Error: 1.508		**Overall Mean No. Meas./Sensor**: 25.3	

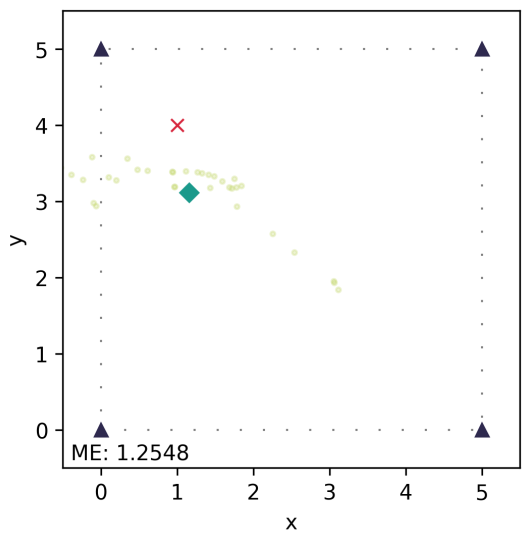
Table 5.5: Results for Experiment 2, Part 1

worsening of the results compared to the first experiment with an overall mean error of 1.508 m, around 0.5 m more than before. While this may be attributed in part to the increase in area spanned by the sensors, the sparsity of the data delivered by the sensors is suspected to be at the root of these issues, given that only ten out of the 32 points evaluated delivered sufficient data to compute a localization in the first place. With an overall average of 25.3 measurements per sensor, the amount of data received was significantly lower than in the first experiment, even when considering only the points that delivered sufficient data for localization. Going from 0.385 m to 2.967 m, the range spanned by the mean errors of the individual results is also significantly higher than in the first experiment.

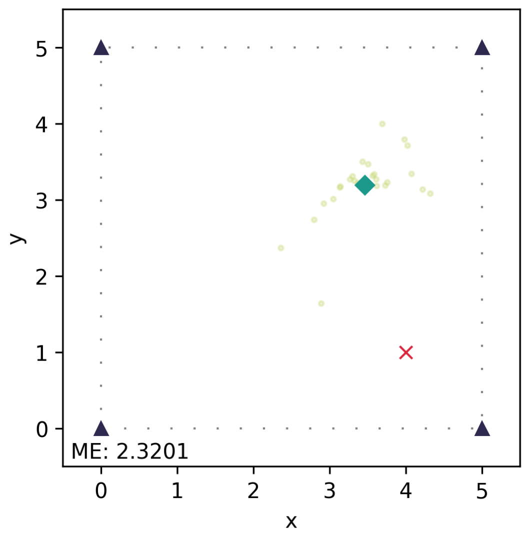
Table 5.6 and Figure 5.6 show the results for the second part of the experiment. It is noticeable that the streaming processing pipeline handles the sparsity of sensor measurements better than the individual evaluations performed in part one of the experiment. This is mostly due to the fact that the processing stream is able to use values for interpolation that lie outside the time intervals defined as resting periods at each point. The results are more accurate than those for the first part of the experiment but worse than those for Experiment 1 with an overall mean error of 1.406 m. It should be noted, however, that the mean error value for point $(4, 1)$ forms an outlier, differing from the next lower value by 1.057 m, more than five times the difference between any other two points (0.198 m). This corresponds to a pattern that was already observed in Experiment 1.



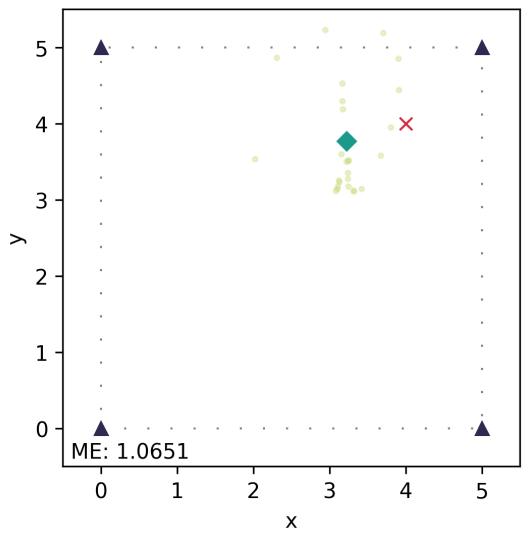
(a) Results for point $(1, 1)$



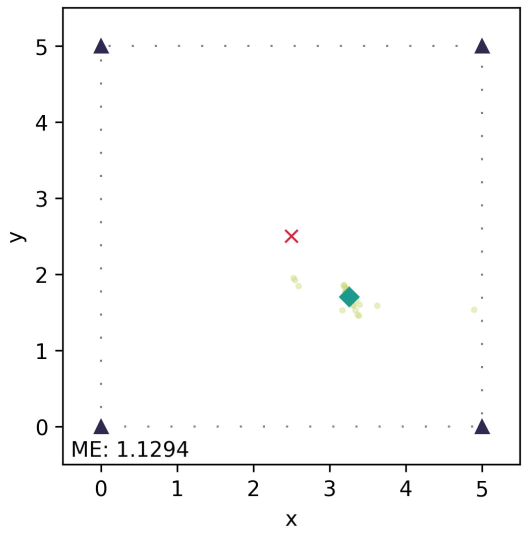
(b) Results for point $(1, 4)$



(c) Results for point $(4, 1)$



(d) Results for point $(4, 4)$



(e) Results for point $(2.5, 2.5)$

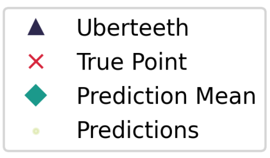


Figure 5.6: The results for Experiment 2, Part 2

True Point (m)	**Mean Estimated Point** (m, rounded)	**Mean Error** (m, rounded)	**No. Localizations**
$(1, 1)$	$(0.989, 1.942)$	1.263	36
$(1, 4)$	$(1.160, 3.108$	1.255	33
$(4, 1)$	$(3.465, 3.188)$	2.320	26
$(4, 4)$	$(3.227, 3.766)$	1.065	26
$(2.5, 2.5)$	$(3.257, 1.703)$	1.129	22
Overall Mean Error: 1.406		**Mean No. Localizations**: 28.6	

Table 5.6: Results for Experiment 2, Part 2

5.3 Experiment 3: Evaluation of Bluetooth Utilization in a Real-World Scenario

5.3.1 Experimental Setup

A Third and final experiment was designed to evaluate the frequency of detectable Bluetooth usage within a random group of people. The experiment was performed at the *Reloading Live* conference, a showcase of how live events can take place in a safe fashion in the context of the SARS-CoV-2 pandemic [49]. Sensors were installed at one of the exponents' stands. This setup is illustrated in Figure 5.7. No further stands were located within a distance of around 3 m around the stand where the experiment was performed. The same Ubertooth One, Asus Tinkerboard combination as in Experiment 2 was used. Due to logistical problems at the site, only three sensors could be used to collect data. Running the full positioning pipeline was, therefore, not possible. Instead, the sensors would simply record all captured packets and compute their LAPs. They could then be analyzed ex-post, using the sensors range as the limit for the detection of a device. The sensors were configured to record over an interval of three hours. To account for non-mobile devices, such as parts of the location's infrastructure, LAPs that appeared over a timespan of more than one hour were filtered out, as were LAPs that were detected for no more than one second.

To validate the number of LAPs detected by the Bluetooth sensors, it was then compared to data collected by a *Xovis* in-store analytics system installed on the premises. It uses *Xovis PC2S* stereo cameras as 3D sensors to detect people and produce statistics, such as dwell or customer frequencies [50]. Among the metrics collected is a count of the people within the range of the camera within discrete time intervals of several lengths, making it ideal for the comparison with the Bluetooth data. The Xovis sensor was attached to the ceiling approximately at the center of the stand.

5.3.2 Results

The number of detected LAPs were aggregated over 15 minute intervals. Figure 5.8 shows a comparison of the number of people counted using the Xovis system to the number of

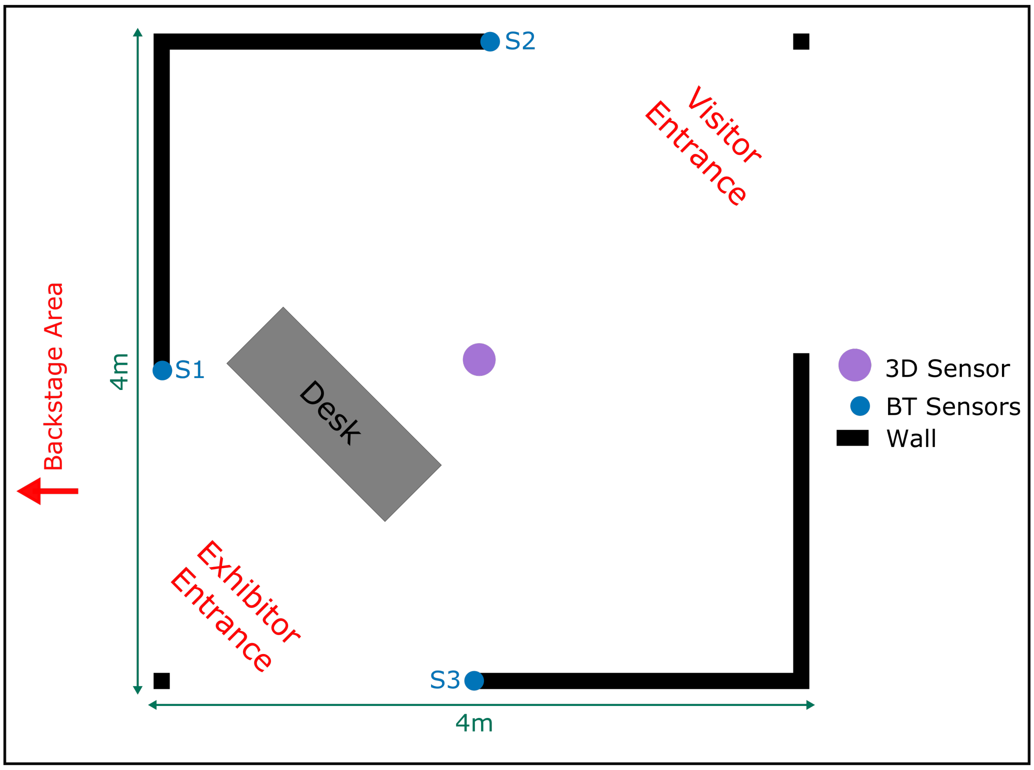
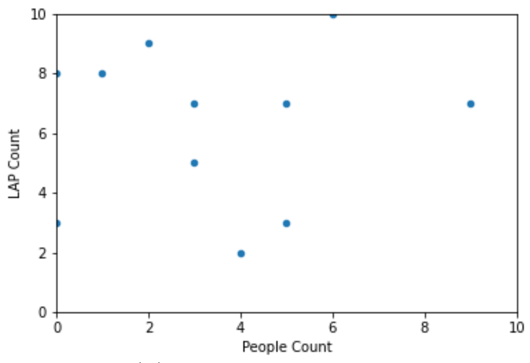


Figure 5.7: The setup for Experiment 3 at a stand at the Reloading Live conference

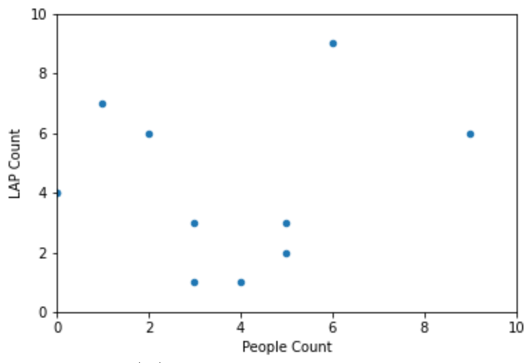
Sensor	**Correlation Coeff.**	**Total No. LAPs Counted**
1	−0.001136	60
2	0.148932	30
3	0.215421	39
merged	0.139848	24
	Total No. People Counted:	38

Table 5.7: Results for Experiment 3

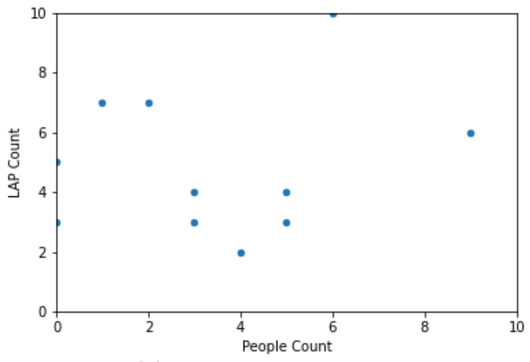
LAPs detected during said intervals. Table 5.7 contains the total number of LAPs counted per sensor and the correlation coefficient between the number of LAPs and number of people counted. For the merged results, a LAP was only counted if it appeared in all three sensors. In summary, little to no correlation exists between the two data sets. This indicates that people at the Reloading Live conference generally did not carry mobile devices with them that were actively using a Bluetooth connection. The total number of LAPs counted diverged from the total number of people counted especially for sensor 1. This may have been because this sensor was closest to the edge of the space used for the conference and therefore the ”backstage” area (located on the left of the stand in Figure 5.7), where people belonging to the conference staff may have caused additional false detections. For the other two sensors, the numbers were closer, especially for sensor 3, which also shows the highest correlation coefficient. One hypothesis which explains this is that sensor 3 was both far away enough from the ”backstage” area to avoid interference caused by the conference personnel, and from the entrance to avoid detecting people who did not actually enter the stand.



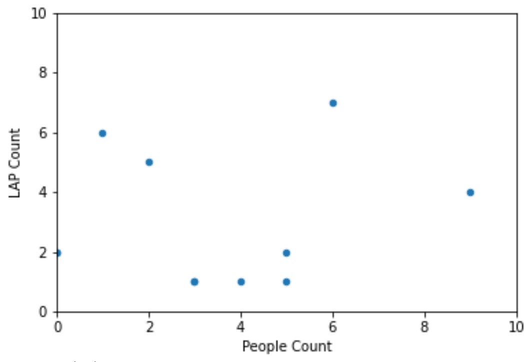
(a) Results for sensor 1



(b) Results for sensor 2



(c) Results for sensor 3



(d) Results for all sensors merged

Figure 5.8: Scatterplot comparing the number of people to the number of LAPs counted over 15 minute intervals in Experiment 3 for each sensor

5.4 Discussion

The three experiments provide a good overview over the performance, the potential and the limitations of the *BluePIL* system. Experiment 1 showed that the device localization method is able to produce results at around 1 m accuracy on average both in indoor and outdoor spaces without the prior calibration of the system to estimate RSS_C and in a completely passive manner, setting it apart from existing approaches. Results for the indoor space were slightly worse, which may be explained by the higher amounts of noise from multipath fading due to the more constrained dimensions. The first part of Experiment 2 did not show the same level of success. The decreased timespan allocated for each measurement combined with the deteriorated performance of the sensors led to large parts of the data being unusable. The performance for the remaining data points was significantly worse than in the first experiment. The second part of the second experiment, however, showed the effectiveness of the streaming architecture, most notably the signal strength merger component, which dealt well with the additional sparsity of signal strength measurements encountered. The localization performance was good, apart from a single outlier, a pattern which was also observed during the first experiment. The results are comparable to existing research, which is pleasing given that the *BluePIL* system estimates locations and channel parameters simultaneously.

The existence of negative outliers in the localization results in both experiments warrants discussion. An inspection of the raw data revealed that these outliers were not caused by a failure at some point in the processing pipeline, but were already present in the data received from the sensors. A plausible explanation for this is that environmental factors, or possibly a combination thereof, may have lead to the RSS values not representing the location of the test subject accurately. The experiments took place in a residential area of Zürich, some of them outdoors. Consequently, influencing factors, such as background noise, the topology of the space, temperature, humidity, etc., could not be controlled and may have led to disturbance in the signal. The test subject carrying the test devices may itself have had an effect on the signal strengths measured, as is suggested by the investigation of the influence of the human body on RSS measurements in [28]. The Ubertooth sensors used present another possible explanation since the accuracy and consistency of the RSS values they deliver is unknown.

The deterioration of sensor performance in Experiment 2 presented a further challenge. Since the environment was identical to the outdoor space used in the first experiment and the frequency of measurements was lower regardless of the distance from the sensor, these two factors do not explain the degradation. One potential explanation could be the change in weather conditions. While the first experiment took place in spring, the second one was performed in mid-summer, with temperatures exceeding 30°C on an asphalt surface. The sensors may have overheated, leading to the decreased frequency and lower accuracy of measurements in the radio components. Existing research suggests a strong influence of operating temperature on RSS measurements for chips very similar to the one used in the Ubertooth One [51]. The unpredictability of the results may have been exacerbated by shade reaching some of the sensors over the duration of the experiment, breaking the assumption that all four sensors exhibit the same radio characteristics. This hypothesis could also explain the better performance in the second part of the experiment, which was executed at a later time on the same day, when temperatures were lower and the entire environment was covered by shade.

Experiment 3 showed that the prevalence of active Bluetooth connections was low in the scenario inspected. This can be explained by the fact that visitors of the conference were seldom actively using their Bluetooth devices (*e.g.* headphones or wearables), since they were engaged in conversation or otherwise occupied. Passively captured Bluetooth packets are, therefore, not a suitable identifier for people in this scenario. Further research is needed to assess the applicability in other scenarios, such as shops, train stations, museums, etc., where a higher rate of usage of Bluetooth devices can be expected.

Comparing the *BluePIL* system to the requirements described in Section 3.2, the following statements can be made:

R1 **The system should be completely passive:** The requirement is satisfied. The system requires no calibration or collaboration with the devices tracked.

R2 **The system should be able to identify individuals based on a Bluetooth signal:** The requirement is satisfied. The system identifies individuals using the LAP from passively captured packets.

R3 **The system should be able to localize individuals based on a Bluetooth signal:** The requirement is satisfied. The system localizes individuals based on passively captured Bluetooth packets and provides localization accuracies between ca. 1 m and 1.4 m on average.

AR1 **The system should be able to work with low-performance hardware:** The requirement is satisfied. The system performs well on the low-cost, low-performance Asus Tinkerboard devices used and did not run into any bottle-necks. More performant sensors may, however, improve the overall accuracy and consistency.

AR2 **The system should be suited for a cloud-based environment:** The requirement is satisfied. The flexible streaming architecture enables the system to move any part of the pipeline to the cloud. As is, it produces data with minimal overhead and, therefore, minimal cost for processing, transmitting and storing in a cloud-based environment.

AR3 **The system should produce results in real time:** The requirement is partially satisfied. The system produces results in near-real-time, optimizing for the trade-off between improvement of the data accuracy and real-time evaluation.

AR4 **The system should be easy to use and require a minimal amount of configuration:** The requirement is satisfied. The only configuration required is the installation of the sensors and the recording of their locations. Everything else is set up automatically.

Chapter 6

Summary and Conclusion

This thesis approaches the problem of passively tracking the identities and locations of Bluetooth devices. An examination of existing research showed that, while a variety of systems exist for both the topic of Bluetooth device identification and device localization, a fully passive method is, so far, lacking. This is the case especially for the area of device localization where approaches based on fingerprinting, as well as those based on a path loss model require prior calibration using a target device. *BluePIL* is a distributed, near-real-time, streaming system that uses a node-sink topology. It defines a data processing pipeline that accomplishes the tasks of identification and localization through passively captured BTBR/EDR packets in several steps, *i.e.* device identification, signal strength filtering, signal strength merging, the localization algorithm and location filtering. *BluePIL* is based on a Python implementation able to run on low-cost hardware and requiring minimal setup, since most configuration is handled automatically. The *BluePIL* system was then evaluated in three experiments. The first of which showed that the device localization method is sound and accurate, producing results with an accuracy of around 1 m in a 12 m^2 area. The second experiment evaluated the system design overall and tested the localization method under more challenging conditions. It showed that the streaming architecture selected for the system is valid and that it is able to localize devices with an accuracy of 1.4 m in a 25 m^2 area. Both experiments showed negative outliers in the localization results that worsened the overall outcome. The cause of these outliers is unknown but may be found in the sensors used or the environment. A third experiment investigated the prevalence of detectable Bluetooth utilization at a trade show and showed that it is not sufficient in order to be used for people counting based on passively captured signals. Overall, the *BluePIL* system fulfills the requirements determined for this thesis and the additional requirements gathered from a concrete, real-world use case.

6.1 Future Work

This thesis produces a variety of possibilities for future research. The system described uses four sensors to accomplish the task of device identification and localization over a limited range. The addition of further sensors could allow for either the extension of said range or for the improvement of the localization accuracy. While the algorithms used for filtering and localization are theoretically capable of working with more than four measurements, approaches to extend the possible range would have to include strategies for the selection of the most useful measurements during the signal strength merger stage. The individual components of the data processing pipeline could be exchanged for alternative methods and compared to the existing system in their effectiveness. This could include sensor fingerprinting approaches based on physical characteristics of the signal for the identification step or location fingerprinting approaches as a localization algorithm.

During the evaluation, a variety of problems with the Ubertooth One sensors used were discovered. Running the system with an alternative sensor could yield information on the source of the outliers discovered in the localization results. In addition, a more precise sensor may enable an improvement of the accuracy of the system overall. The repetition of Experiments 1 and 2 in a more controllable environment could deliver further insights into the performance of *BluePIL*. The indoor and outdoor spaces used in the evaluation where somewhat limited in this regard, being constrained in topology and exhibiting a large amount of background noise and environmental factors, that may have influenced the results. A large indoor space, such as an underground parking garage or a gymnasium, may provide more isolation.

Finally, a repetition of Experiment 3 in different scenarios would provide insight into the prevalence of detectable Bluetooth utilization in other environments than the trade show examined. Places where a higher amount of Bluetooth usage is expected, such as a train station or a study room, would be especially interesting.

Bibliography

[1] Bluetooth SIG, "Market update 2020," Bluetooth SIG, 2020. [Online]. Available: https://www.bluetooth.com/bluetooth-resources/2020-bmu/ (Accessed 20.3.2020).

[2] Livealytics, "Livealytics - we make live experience measurable," https://www.livealytics.com, 2020, accessed: 2020-07-26.

[3] B. B. Rodrigues, "Inosuisse funds cooperation between the university of zurich and livealytics," https://www.csg.uzh.ch/csg/en/news/PasWITS-Research-Project0.html, Mar. 2020, accessed: 2020-07-24.

[4] *Bluetooth core specification v5.2*, Bluetooth SIG, Dec. 2019.

[5] Bluetooth SIG, "Vision and mission," https://www.bluetooth.com/about-us/vision/, 2020, accessed: 2020-07-17.

[6] IEEE, "Ieee standard for local and metropolitan area networks: Overview and architecture," *IEEE Std 802-2014 (Revision to IEEE Std 802-2001)*, pp. 1–74, 2014.

[7] Great Scott Gadgets, "Ubertooth," https://github.com/greatscottgadgets/ubertooth, 2020, revision: c2cc373, accessed: 2020-07-19.

[8] G. Carle and C. Schmitt, Eds., *Proceedings of the Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM), Winter Semester 12/13*, ser. Network Architectures and Services (NET). Munich, Germany: Chair for Network Architectures and Services, Department of Computer Science, Technische Universität München, Feb. 2013, vol. NET-2013-02-1.

[9] Hacker Warehouse, "Ubertooth one," https://hackerwarehouse.com/product/ubertooth-one/, 2020, accessed: 2020-08-18.

[10] V. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello, "Bayesian filtering for location estimation," *IEEE pervasive computing*, vol. 2, no. 3, pp. 24–33, 2003.

[11] R. Labbe, "Kalman and bayesian filters in python," https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python, commit 91f8010bee8cd7e07bdca338ecd90c3dd7735e92, 2014.

[12] Y. Kim and H. Bang, "Introduction to kalman filter and its applications," in *Introduction and Implementations of the Kalman Filter.* IntechOpen, 2018.

[13] G. Li, E. Geng, Z. Ye, Y. Xu, J. Lin, and Y. Pang, "Indoor positioning algorithm based on the improved rssi distance model," *Sensors*, vol. 18, no. 9, p. 2820, 2018.

[14] J. B. Andersen, T. S. Rappaport, and S. Yoshida, "Propagation measurements and models for wireless communications channels," *IEEE Communications Magazine*, vol. 33, no. 1, pp. 42–49, 1995.

[15] M. Ryan, "Bluetooth: With low energy comes low security," in *Presented as part of the 7th {USENIX} Workshop on Offensive Technologies*, 2013.

[16] J. K. Becker, D. Li, and D. Starobinski, "Tracking anonymized bluetooth devices," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 50–65, 2019.

[17] C. Zuo, H. Wen, Z. Lin, and Y. Zhang, "Automatic fingerprinting of vulnerable ble iot devices with static uuids from mobile apps," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1469–1483.

[18] J. Huang, W. Albazrqaoe, and G. Xing, "Blueid: A practical system for bluetooth device identification," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 2849–2857.

[19] O. Whitehouse, "War nibbling: Bluetooth insecurity," *white paper, stake Inc., Oct*, 2003.

[20] M. Herfurt and C. Mulliner, "Blueprinting: Remote device identification based on bluetooth fingerprinting techniques," in *21st Chaos Communication Congress (21C3)*, 2004.

[21] D. Spill and A. Bittau, "Bluesniff: Eve meets alice and bluetooth." *WOOT*, vol. 7, pp. 1–10, 2007.

[22] R. Faragher and R. Harle, "Location fingerprinting with bluetooth low energy beacons," *IEEE journal on Selected Areas in Communications*, vol. 33, no. 11, pp. 2418–2428, 2015.

[23] A. Bekkelien, M. Deriaz, and S. Marchand-Maillet, "Bluetooth indoor positioning," Master's thesis, University of Geneva, 2012.

[24] Y. Zhuang, J. Yang, Y. Li, L. Qi, and N. El-Sheimy, "Smartphone-based indoor localization with bluetooth low energy beacons," *Sensors*, vol. 16, no. 5, p. 596, 2016.

[25] S. Chai, R. An, and Z. Du, "An indoor positioning algorithm using bluetooth low energy rssi," in *2016 International Conference on Advanced Materials Science and Environmental Engineering*. Atlantis Press, 2016.

[26] J.-H. Huh and K. Seo, "An indoor location-based control system using bluetooth beacons for iot systems," *Sensors*, vol. 17, no. 12, p. 2917, 2017.

[27] P. Dickinson, G. Cielniak, O. Szymanezyk, and M. Mannion, "Indoor positioning of shoppers using a network of bluetooth low energy beacons," in *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 2016, pp. 1–8.

[28] Y. Wang, X. Yang, Y. Zhao, Y. Liu, and L. Cuthbert, “Bluetooth positioning using rssi and triangulation methods,” in *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*. IEEE, 2013, pp. 837–842.

[29] J. Du, J.-F. Diouris, and Y. Wang, “A rssi-based parameter tracking strategy for constrained position localization,” *EURASIP Journal on Advances in Signal Processing*, vol. 2017, no. 1, pp. 1–10, 2017.

[30] M. Versichele, L. De Groote, M. C. Bouuaert, T. Neutens, I. Moerman, and N. Van de Weghe, “Pattern mining in tourist attraction visits through association rule learning on bluetooth tracking data: A case study of ghent, belgium,” *Tourism Management*, vol. 44, pp. 67–81, 2014.

[31] A. Alhamoud, A. A. Nair, C. Gottron, D. Böhnstedt, and R. Steinmetz, “Presence detection, identification and tracking in smart homes utilizing bluetooth enabled smartphones,” in *39th Annual IEEE Conference on Local Computer Networks Workshops*. IEEE, 2014, pp. 784–789.

[32] L. Schauer, M. Werner, and P. Marcus, “Estimating crowd densities and pedestrian flows using wi-fi and bluetooth,” in *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2014, pp. 171–177.

[33] DP-3T, “Dp3t - decentralized privacy-preserving proximity tracing,” https://github.com/DP-3T/documents, 2020, accessed: 2020-07-27.

[34] Google, “Covid-19-benachrichtigungen: Wie wir die gesundheitsbehörden durch technologie bei der eindämmung von covid-19 unterstützen,” https://www.google.com/covid19/exposurenotifications/, 2020, accessed: 2020-07-27.

[35] J. Brunner, “Optimizing the collection and processing of wi-fi probe requests,” Communication Systems Group – University of Zurich, Independent Study, Jun. 2020.

[36] International Data Corporation (IDC), “Smartphone market share,” https://www.idc.com/promo/smartphone-market-share/vendor, 2020, accessed: 2020-07-30.

[37] M. I. Lourakis *et al.*, “A brief description of the levenberg-marquardt algorithm implemented by levmar,” *Foundation of Research and Technology*, vol. 4, no. 1, pp. 1–6, 2005.

[38] K. Madsen, H. Nielsen, and O. Tingleff, “Methods for non-linear least squares problems (2nd ed.),” p. 60, Jan. 2004.

[39] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.

[40] Asus, “tinker board – tinker your way to the future,” https://www.asus.com/us/Single-Board-Computer/Tinker-Board/, 2020, accessed: 2020-08-04.

[41] armbian, “armbian – linux for arm development boards,” https://www.armbian.com, 2020.

[42] Apple, "Macbook pro (13-inch, 2017, four thunderbolt 3 ports) – technical specifications," https://support.apple.com/kb/sp755?locale=en_US, 2020, accessed: 2020-08-04.

[43] M. Rocklin, "Streamz," https://streamz.readthedocs.io/en/latest/, 2020, revision de166a40, accessed: 2020-08-04.

[44] R. Holeman, "Pyubertooth," https://github.com/hackgnar/pyubertooth, 2020, revision 080bc6f, accessed: 2020-08-04.

[45] Python Software Foundation, "asyncio – asynchronous i/o," https://docs.python.org/dev/library/asyncio.html, 2020, version 3.10.0a0, accessed: 2020-08-04.

[46] Great Scott Gadgets, "libbtbb," https://github.com/greatscottgadgets/libbtbb, 2020, revision: 6c7b9ff, accessed: 2020-07-19.

[47] The SciPy community, "Scipy.org – documentation," https://www.scipy.org/docs.html, 2020, accessed: 2020-08-04.

[48] R. R. Labbe, "Filterpy," https://asyncio.readthedocs.io/en/latest/index.html, 2020, version 1.4.4, accessed: 2020-08-04.

[49] MCH Group, "Reloading live: Experiencing the live event of the future," https://www.scipy.org/docs.html, 2020, accessed: 2020-08-11.

[50] Xovis, "Pc2s," https://www.xovis.com/en/products/detail/pc2s/, 2020, accessed: 2020-08-11.

[51] C. A. Boano, N. Tsiftes, T. Voigt, J. Brown, and U. Roedig, "The impact of temperature on outdoor industrial sensornet applications," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 3, pp. 451–459, 2009.

Abbreviations

AA	Access Address
AC	Access Code
BTBR	Bluetooth Basic Rate
BTEDR	Bluetooth Enhanced Data Rate
BTLE	Bluetooth Low Energy
CAC	Channel Access Code
CID	Company Identifier
CRC	Cyclic Redundancy Check
EUI-48	48-bit Extended Unique Identifier
FHSS	Frequency Hopping Spread Spectrum
GFSK	Gaussian Frequency-Shift Keying
HCI	Host Controller Interface
HEC	Header Error Code
IoT	Internet of Things
IP	Internet Protocol
ISM	Industrial, Scientific and Medical
KPI	Key Performance Indicator
LAP	Lower Address Part
LFSR	Linear-Feedback Shift Register
LM	Levenberg-Marquardt
LSB	Least Significant Bit
MAC	Media Access Control
MSB	Most Significant Bit
NN	Neural Network
OUI	Organizationally Unique Identifier
RSSI	Received Signal Strength Indicator
SBC	Single Board Computer
SIG	Special Interest Group
SoC	System on Chip
TCP	Transmission Control Protocol
TDD	Time Division Duplex
USB	Universal Serial Bus
UUID	Universally Unique Identifier

List of Figures

List of Tables

Appendix A

Contents of the Repository

The code repository is structured as follows:

- Root Directory: Contains Python scripts for running the node and the sink applications, the *BluePIL* configuration file, and the requirements file for the applications' dependencies
- `data` Directory: Contains the raw data collected during the experiments
- `data_analysis` Directory: Contains the code used to analyze the experimental data
- `node` Directory: Contains the code for the *BluePIL* node application
- `sink` Directory: Contains the code for the *BluePIL* sink application
- `node_setup`: Contains scripts and files for the setup of the node application on an Asus Tinkerboard running Armbian

Appendix B

Installation Guidelines

B.1 Installation of Dependencies

BluePIL requires Python v3.8 to be installed. A virtual environment capable of executing both the host and the sink code can then be installed by running the following set of commands

```
python3 -m venv env
source env/bin/activate
pip install -r requirements.txt
```

Dependencies for the BluePIL node code to run on an Asus Tinkerboard running Armbian can be installed by running the script `node_setup/install.sh` on the device. Note that an Ubertooth One sensor must be connected to each one of the Tinkerboards via USB.

B.2 Running the Application

The *BluePIL* node application can be started by running `python run_node.py`. The sink application can be started by running `python run_sink.py`. In order for the communication between node and sink to work properly, the node application has to be started on each node before starting the sink application.

The application can be configured using the file `bp.json`. The following options are available for the parameter `mode`:

- `RAW`: Configures the nodes to collect raw data without running the positioning pipeline and stores it on the sink

- `RAW_LOCAL`: Configures the nodes to collect raw data without running the positioning pipeline and stores it on the nodes. In this mode, the nodes may run detached from the sink and the sink application exits after sending a start command to the nodes. It may then be run again to stop the execution.
- `POSITIONING`: runs the full positioning pipeline and stores the results on the sink

In addition, for each node (`node1-4`), the following parameters must be configured:

- `ip`: The IP address of the node
- `loc`: The location of the node in x and y coordinates