UCLouvain

École polytechnique de Louvain

Radio activity optimisation for Wireless Personal Communications

A Bluetooth Low Energy application

Author: **Jérome EERTMANS**
Supervisors: **Jérôme LOUVEAUX, Claude OESTGES**
Advisor: **Sam GEERAERTS**
Reader: **Mathieu XHONNEUX**
Academic year 2020–2021
Master [120] in Electro-mechanical Engineering

Abstract

Similarly to other technologies, Bluetooth has received increasing interest over the past 20 years. In particular, the Bluetooth Low Energy (BLE) market undergoes constant growth for more than a decade, with billions of device units shipped every year. Initially thought to be simple and low power, BLE is getting more complex year after year as new applications emerge. The state-of-the-art products require multi-connectivity and high-quality link while staying low power. This leverage some severe design challenges that engineers must solve. After highlighting the complexity related to modern challenges, this document proposes a novel algorithm for multiple connections tasks scheduling. Another very challenging mission is coexistence interference. BLE uses the same frequency bands as Wi-Fi, a prevalent technology in most homes, companies and public facilities. As such, channel assessment, a mechanism to evaluate the quality of the different frequency bands, is essential to avoid interference. Current implementation sees Wi-Fi as the source of interference, but not the other way around. An enhanced channel assessment is proposed to detect the strongest Wi-Fi signal disregarding its signal strength and the number of active Wi-Fi channels. After detection, the BLE channels overlapping with Wi-Fi's are removed from the channel map to avoid interference. Finally, the two contributions discussed reveal encouraging results for real-case applications. Task scheduling always finds an excellent solution in less than 100 ms. Under several fairly realistic assumptions, this time can be reduced to under 10 ms, or even 1 ms in the best cases. As for Wi-Fi signal detection, this improvement adds less than 20 µs to the current execution of the Channel Assessment. This method, which is robust to outliers, makes it possible to detect, in a few milliseconds, the location of the strongest Wi-Fi channel in a crowded spectrum with a minimum accuracy of between 2 MHz and 10 MHz depending on the case.

Acknowledgements

The writing this thesis would not have been possible without the help of several people, and I wish to acknowledge them.

Firstly, I would like to thank my two thesis supervisors, Pr. Claude Oestges and Pr. Jérôme Louveaux, for their continuous help, support and advice throughout this academic year.

Then, I am particularly grateful to my thesis advisor, Sam Geeraerts, for all the knowledge he shared with me. I am thankful for his assistance on the technical points as well as his review of my work.

Furthermore, I wish to express my gratitude to all the contributors of the *Tikz* and *PGFplots* TEX packages for the work they have achieved but also for the help they gave me in answering my questions. If my thesis looks like this, it is mainly thanks to their work to create high-quality and open-source graphical tools.

Finally, I would like to thank my parents for helping me proofreading the English grammar of my document.

Acronyms

ACK Acknowledgement
ACL Asynchronous Connection-Less
AES Advanced Encryption Standard
AFH Adaptive Frequency Hopping
App Application Layer
ATT Attribute Protocol

BER Bit Error Rate
BIS Broadcast Isochronous Stream
BLE Bluetooth Low Energy

CA Channel Assessment
CAGR Compound Annual Growth Rate
CCA Clear Channel Assessment
CCM Cipher Block Chaining
CIS Connected Isochronous Stream
CRC Cyclic Redundancy Check

DSSS Direct-Sequence Spread Spectrum

ECC Error Correction Code

GAP Generic Access Profile
GATT Generic Attribute Protocol
GCD Greatest Common Divisor

IFS Inter Frame Space
IID Independent and Identically Distributed
IIR Infinite Impulse Response
IoT Internet of Things
ISM Industrial, Scientific and Medical

L2CAP Logical Link Control & Adaptation Protocol
LCM Lowest Common Multiple
LE Low Energy
LL Link Layer

MIC Message Integrity Check

NACK No Acknowledgement
NESN Next Expected Sequence Number
NOCA No Channel Assessment

OFDM Orthogonal Frequency-Division Multiplexing

PDU Protocol Data Unit
PER Packet Error Rate
PHY Physical Layer
ppm parts per million
PRNG Pseudo-Random Number Generator

RSSI Received Signal Strength Indicator
RV Random Variables

SMP Security Manager
SN Sequence Number
SNR Signal to Noise Ratio
SSBR Signal to Second Biggest Ratio

Symbols

Symbol	Description	Unit
b	Bit shorthand symbol	bit
B	Byte shorthand symbol	byte
$\lceil \cdot \rceil$	Ceil rounding	-
t	Continuous time variable	s
$f(t)$	Continuous function f with parameter t	-
$*$	Convolution operator	-
$f[n]$	Discrete function f parameter n – also called index	-
i, j	Discrete indices, often used to differentiate parameter values	-
n	Discrete time variable or an integer variable	-
ρ	Drift, relative rate often expressed in ppm (1×10^{-6})	-
$\lfloor \cdot \rfloor$	Floor rounding, often used to denote integer division	-
f	Frequency	Hz
Q_N	Nominal – or ideal – value for quantity Q	same as Q
$\mathcal{N}\left(\mu, \sigma^2\right)$	Normal distribution with mean μ and variance σ^2	-
T	Period, interval	s
$\ll$ ($\gg$)	Smaller (bigger), by order(s) of magnitude	-
$\sum_{j \neq i}$	Summation over all j values such that $j \neq i$	-
$\sum_{i, j > i}$	Summation over all (i, j) pairs such that $j > i$	-
$\mathcal{Z}\{x[n]\}$	Z-transform of function $x[n]$, sometimes directly written as $X(z)$	-

Contents

1 Introduction

In the last decades, the market of connected devices has been steadily growing, bringing almost every day new applications and raising a set of challenges that are to be solved. As the number of applications grows, so does the complexity of implementing solutions to the problems. Smartphones, which were only capable of handling a single Bluetooth connection a few years ago, can simultaneously connect to multiple wireless devices.

The multiple concurrent connections paradigm, which was not necessarily present before, is one of the many examples of engineers' missions when designing new connected devices.

The Bluetooth Low Energy (BLE) technology is an excellent example in terms of market growth and complexity. This protocol started as a straightforward and low power solution and now ends in most connected devices as its popularity rapidly increased. In partnership with NXP Semiconductors company, and more specifically their headquarters of the Personal Health product line based in Leuven, Belgium, a study of some state-of-the-art challenges was conducted. Therefore, a significant part of this document is dedicated to reviewing the BLE technology and its evolution. It also contains practical experiments, measurements, and algorithm to address some of the challenging tasks raised throughout the present study. The objective of this thesis is threefold: *(i)* give a thorough introduction to BLE to provide the necessary material to understand the different parameters state-of-the-art products rely upon. *(ii)* Emphasise the emerging challenges and their inherent complexity. *(iii)* Develop enhanced solutions and measure their performance impact. The content of this thesis is therefore organised as follows.

First, it introduces, in a higher to lower level fashion, the building blocks of the BLE technology and some of its key aspects in Chapter 2, highly inspired by Robin Heydon's book [1]. It also highlights the advancements relevant to this thesis. The role of this Chapter is not to cover every aspect of BLE but rather to provide the necessary information to appreciate the content of the following Chapters. Then, Chapter 3 compares the initial objectives of the technology with the state of the art of the market in term of BLE-capable devices: specifications have changed since Heydon's book, and new paradigms have arisen from them. Some modern challenges are discussed, and Chapter 4 presents several contributions that aim to enhance performance on two aspects: communication tasks scheduling and better interference avoidance. Next, Chapter 5 provides several quantitative analyses of how well these algorithms work, as well as other aspects presented earlier to evaluate the impact of specific device parameters. To this end, a series of realistic-case experiments and simulations have been carried out, and the measurements are discussed to compare theory and practice and validate models. Finally, Chapter 6 concludes this thesis by synthesising the results and providing some ideas for further researches.

The world of wireless personal networks is multidisciplinary. It results from a fruitful collaboration of electrical, computer sciences, and other fields. In many electronic applications, devices design requires both hardware and software optimisation. As such, this document gives prominence to answering the questions on both electrical and computer sciences backgrounds.

The writing of this document was intended to be understandable from any scientific, with some basis in engineering, electrical and programming domains, without any prior knowledge in Bluetooth communications. For some specific topics, the Appendices will provide additional help and details.

2 Bluetooth Low Energy

2.1 Historical background and motivation

In the late 1990s, many technologies still used today were introduced, such as Wi-Fi, Bluetooth, and ZigBee. Since then, they have evolved in different ways, and Figure 2.1 depicts the growth in speed for Bluetooth and Wi-Fi protocols. In 2010, BLE was created with the idea that it must be low power, cheap, and straightforward. Both BLE and Bluetooth Classic evolve in parallel even though they share the same version names.

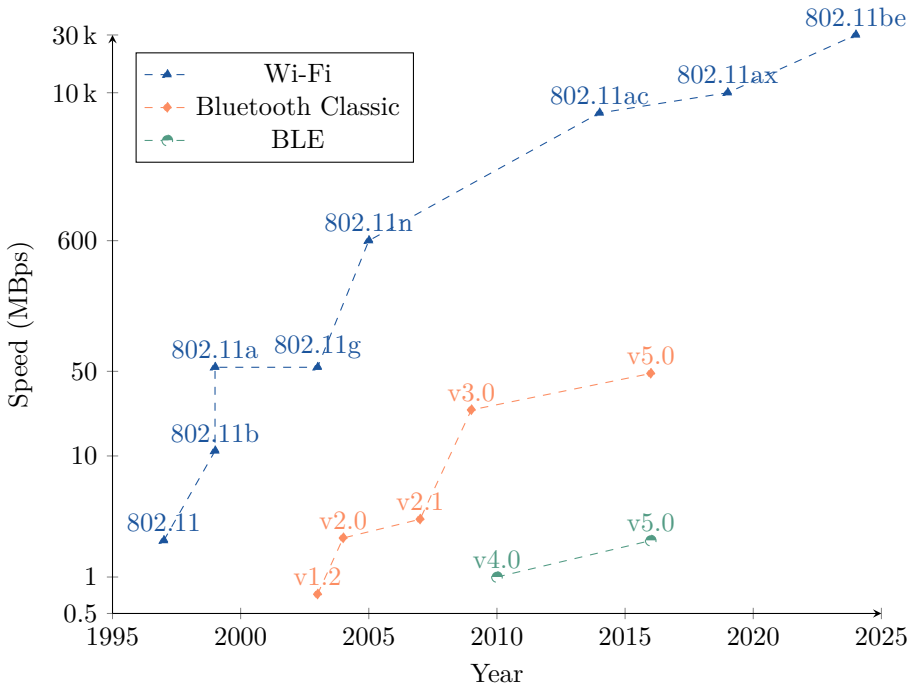


Figure 2.1: Comparison of the evolution of the speed for different technologies [2, 3].

Because these two Bluetooth technologies are very close to each other, many devices allow compatibility between them [1, p. 6]. They both work in the 2.4 GHz Industrial, Scientific and Medical (ISM) band, a public portion of the radio spectrum shared by many other protocols. Those similarities allow device manufacturers to cheaply build BLE-ready products when they already supported the Bluetooth Classic. Table 2.1 summarises

the main differences between the two technologies, highlighting the low power advantage that the BLE has compared to Bluetooth Classic.

Specifications	Bluetooth Classic	BLE v4.2
Range	100 m	>100 m
Data Rate	1 − 3 Mbps	1 Mbps
Application Throughput	0.7 − 2.1 Mbps	0.27 Mbps
Latency	100 ms	6 ms
Voice capable	Yes	No
Power consumption	1 W	0.01 − 0.5 W
Peak current consumption	<30 mA	<15 mA

TABLE 2.1: Non exhaustive comparison between BLE and Bluetooth classic [4].

Even though the Bluetooth Classic was already quite popular, the BLE release widened the Bluetooth market to many more connected devices, such as the Internet of Things (IoT), thanks to its very low power nature. In 2020, see FIGURE 2.2, the Bluetooth Marked Update predicted that, by 2024, 100% of new platform devices would support both technologies (dual-mode) and 35% of annual shipments would be Low Energy (LE) single-mode [5, p. 10]. Introduced in version 5.2 of the Bluetooth Core Specification, LE Audio is foreseen to be the next generation of Bluetooth audio.

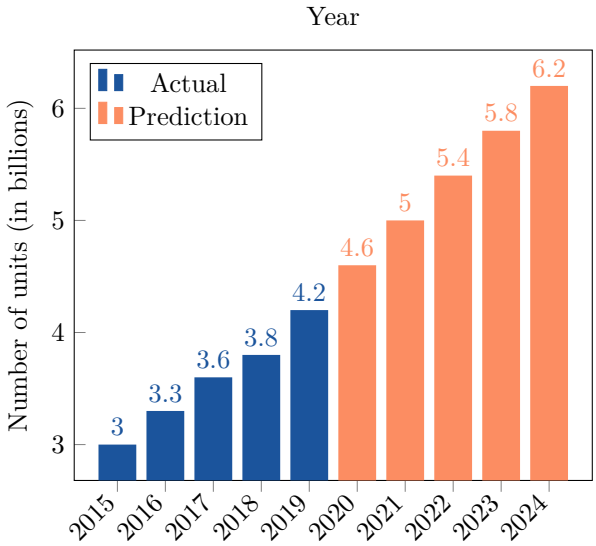


FIGURE 2.2: Number of Bluetooth devices shipped per year [5, p. 9].

With a predicted Compound Annual Growth Rate (CAGR) of 8% for the next five years and the importance of lower power products, BLE technologies have a bright future awaiting them, with many exciting challenges that engineers will have to solve, and some of these will be further discussed in this thesis.

2.2 2.4 GHz ISM BAND

2.2.1 BLE in the ISM band

There are numerous reasons how BLE technologies achieve low-power communications at a relatively low cost. One of which is the use of the 2.4 GHz ISM band. Not only is this portion of the radio spectrum unlicensed, but

it is also available in most countries. This advantage gives device manufacturers the ability to design products that can be sold worldwide without dealing with different frequency bands depending on local restrictions. Nonetheless, one downside with this approach is that it must also be shared among all the technologies that would like to use it: Bluetooth Classic, Wi-Fi, ZigBee, etc.

Given that the spectrum has no restriction on who can use it, each protocol has its way of sub-dividing the spectrum (see FIGURE 2.3). Sharing the same frequency bands, of course, creates interference, also called coexistence, between devices and must be taken into account when designing them.

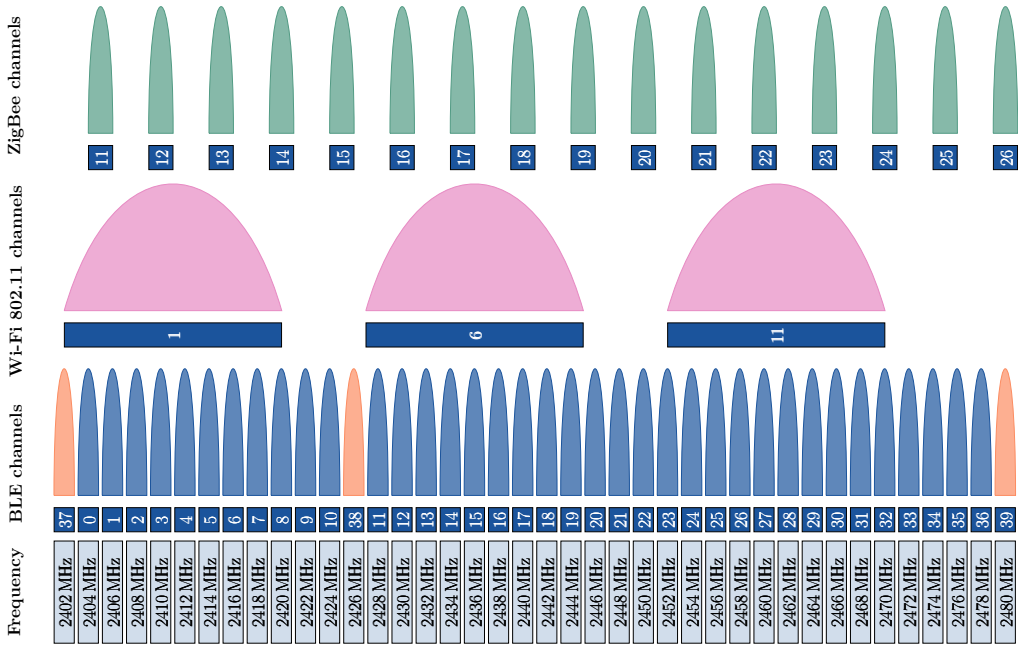


FIGURE 2.3: BLE and other popular technologies in 2.4 GHz ISM band, inspired from [6, Figure 4] and [7, Figure 7]. Relevant channel numbers are indicated under each spectrum lobe and the length of the lobes are proportional to their bandwidth.

BLE device design procedure is also very dependent on the environment the device will be in. Home-theatres with Bluetooth audio pairing will mostly encounter a Wi-Fi signal from the same house and maybe from the neighbours if one lives in an apartment. In contrast, wireless audio earbuds will sometimes be used indoors in a very static environment, but also outdoors where geometries and weather can vary a lot (*e.g.*, cities, rural environments, and forests).

Another downside of this portion of the spectrum is its poor propagation properties in water. Since humans are mostly made of water, the attenuation of a signal getting through someone's body can be very high. In 2018, a paper [8] analysed the shadowing impact of a human body on wireless communications at 2.4 GHz. They concluded that the worst-case scenario happens when at least one of the antennas is very close to the body: the path loss can be increased by up to 20 dB in such cases. Moreover, this scenario happens in many real cases, such as when someone is listening to music using wireless earbuds connected to his phone, located in his pocket, see FIGURE 2.4.

FIGURE 2.4: Illustration of the wireless propagation through human body problem: here, in the worst-case scenario, the smartphone located in the right pocket must communicate with the left earbud, meaning that the signal has to get through about 70 cm of the human body for a typical 1.8 m tall person. This image is an adapted version of the *Man Walking Cartoon Vector* provided by VideoPlasty under CC BY-SA 4.0 license.

Even though the model presented in FIGURE 2.4 is not the same as the one studied in [8], a relatively safe assumption can be that the shadowing impact of the human body will not be negligible in both cases. Indeed, the former assumes a vertical distance through the body of 70 cm while [8] models a maximum horizontal distance of 31 cm. The difference between the transmit and the received power is called path loss: it is the consequence of multiple complex interactions between the signal and the environment. It is often simplified to be mainly linked to the distance between the two communicating devices, as the wavefronts have a decreasing power density per area when the signal gets further away from its source, see FIGURE 2.5.

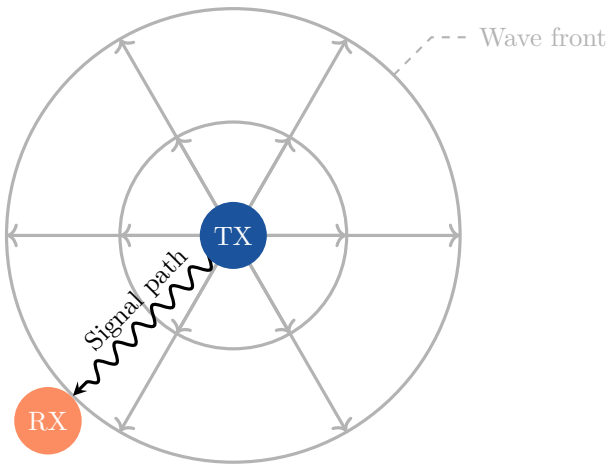


FIGURE 2.5: Expanding wavefronts of a radio signal, inspired from [9, Figure 6].

Radio receivers are always sensitive to the signal power, expressed in dB, and thus have a preferred received signal strength range [9], called the golden range. If the signal power falls out of this range, the packet data

recovery may not work as expected and lead to errors. Also, a too-small power would often mean that the signal becomes indistinguishable from the background noise. A too strong signal may interfere severely with other communications while wasting unnecessary energy. Control of the transmit power could therefore be a solution to those problems.

To this end, as of version 5.2, BLE comes with a LE Power Control feature. The latter allows devices to communicate the transmit signal strength and ask for power control to keep received signals inside the golden range. This feature uses the already existing Received Signal Strength Indicator (RSSI) from each channel to build up an estimate of the received power from a given signal. It computes the path loss using the transmit power specified in the LE Power Control communications, minus the received power. Power control improves the user experience, the device overall power consumption, and the coexistence in the ISM band, benefiting all the devices using those frequencies.

2.2.2 About Wi-Fi

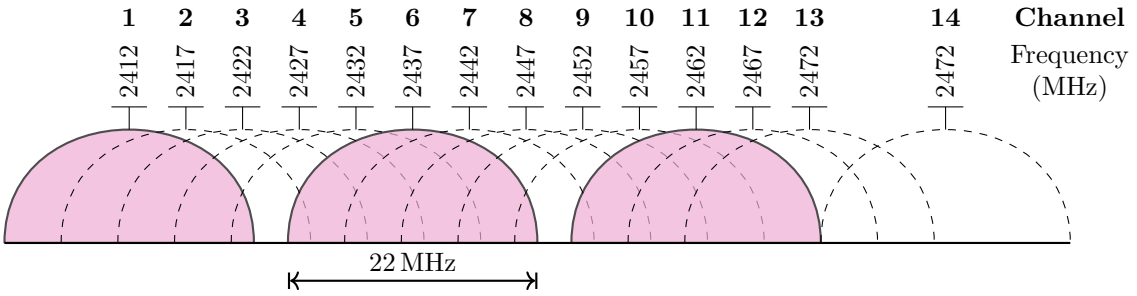


Figure 2.6: 802.11b Wi-Fi channels within the 2.4 GHz ISM band, inspired from [10].

Contrary to what can be thought by observing Figure 2.3, the Wi-Fi protocol can use other channels than channels 1, 6, and 11. Within the 2.4 GHz ISM band are located 14 different Wi-Fi channels that can overlap, see Figure 2.6. In Europe, their width can span from 20 MHz to 40 MHz [10]. Prior versions used Direct-Sequence Spread Spectrum (DSSS) modulation and recent versions use Orthogonal Frequency-Division Multiplexing (OFDM) modulation or similar. While the spectrum shape obtained with DSSS is very close to the one illustrated in Figure 2.3, the shape obtained with OFDM modulation is much flatter – resulting in an almost constant spectrum over a channel, see Figure 2.7.

(a) DSSS – 22 MHz width (b) OFDM – 20 MHz width

Figure 2.7: Comparison between DSSS and OFDM spectra.

2.3 Communication modes and security

Now that the communication medium, *i.e.*, the 2.4 GHz ISM band, and the challenge of sending a message through this medium have been introduced, this Section shortly describes the basic structure of BLE devices, the main communication modes and how packet security is achieved.

2.3.1 BLE structure

The structure of most BLE devices can be split into three stacks, as in FIGURE 2.8. The Physical Layer (PHY) contains the analogue circuitry required for communicating with other devices. The Link Layer (LL), detailed in the next SECTION, directly interacts with the PHY and is responsible for advertising, scanning, and creating and maintaining connections. Together, they constitute the Controller. Given that they have strict time requirements, they often have dedicated hardware and are separated from the other stacks.

Next, the Host and Application handle the part of BLE that is more software related. They will not be detailed here, but Robin Heydon's book contains more information on this topic.

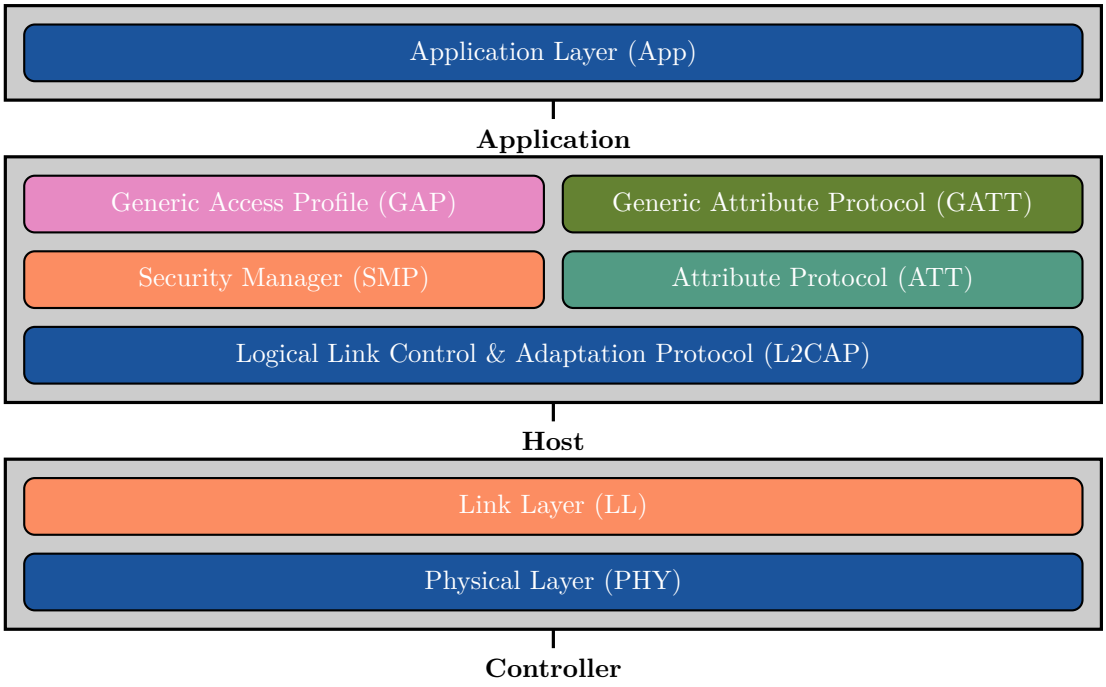


FIGURE 2.8: Basic BLE structure from lower – hardware – (south) to higher – software – (north) level.

Both the Host and the Controller have different sets of protocols for communicating between the various endpoints in the architecture. Among those, the Asynchronous Connection-Less (ACL) protocol is the standard radio link for data packets. The other protocols are not relevant to this document.

In short, ACL is a communication protocol that is used to transmit data communication. An ACL packet is made of four blocks [11]: *(i)* the access code, *(ii)* the packet header, *(iii)* the payload, *i.e.*, the data content, and *(iv)* the Cyclic Redundancy Check (CRC) to detect errors. This kind of link is used when message integrity is preferred over latency.

2.3.2 Link Layer

The BLE Link Layer can be described as a finite-state-machine, and any BLE device must be in any one of the following states:

- Advertising;
- Scanning;
- Initiating;
- Connected;
- Isochronous Broadcasting;
- Synchronisation
- or Standby.

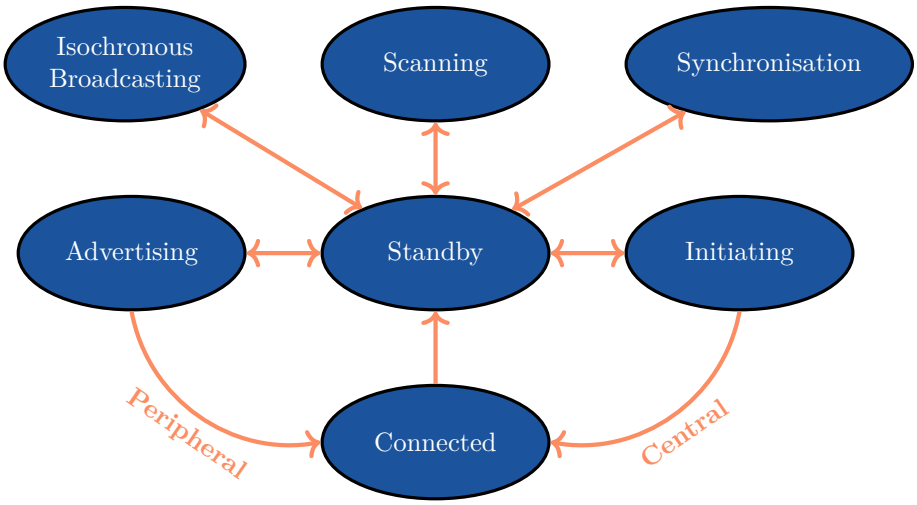


FIGURE 2.9: Link Layer internal finite state machine.

As described in FIGURE 2.9, not all states have a direct connection with each other. Some states require to go through two or more steps to be reached, and this is the case with the Connected state: a device can only enter this state after Advertising or Initiating. This previous state will define the role of the device during the connection.

One crucial aspect of BLE connections is the asymmetry between the possible roles that devices can occupy. There are two roles, central (previously known as Master) and peripheral (Slave), and they have a very different way of behaving. As its name implies, the central will be at the centre of the communication and decide everything: the timing, channels to use, and all the rest. As such, there can only be one central for each connection.

The peripheral, on the other hand, must follow what the central says. If it wants to change the connection parameters, it may ask the central for it, but it has no guarantee that the latter will accept its request. Simple connections only contain one peripheral, but it is possible, as described in SECTION 2.3.3, to have configurations where one central communicates to multiple peripherals in the same connection.

Notation For conciseness, this document will sometimes abbreviate central devices with the letter C. The peripherals with the letter P. In a network diagram, see example in FIGURE 2.10, a line between two devices represents a connection, where the direction of the arrow indicates the role of the device in this connection: it points towards the peripheral. This notation will become helpful when the same device is implied in multiple connections and shares different roles.

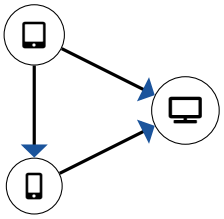


FIGURE 2.10: Central-peripheral notation example. The TV is the peripheral of two connections: one with the phone (below) and another one with the tablet (above). Then, the phone is also the peripheral of a connection with the tablet.

Finally, the detailed structure of a LL packet is displayed in FIGURE 2.11. Prior to v5, packets were limited to a Protocol Data Unit (PDU) of 39 B. In BLE v5, the PDU was changed to allow for larger payloads, increasing the PDU size up to 257 B. One other important change was the extension of the preamble to let

the device select the bitrate. A simplified comparison of the differences between pre-v5 and v5 is done further below with FIGURES 2.13 and 2.14.

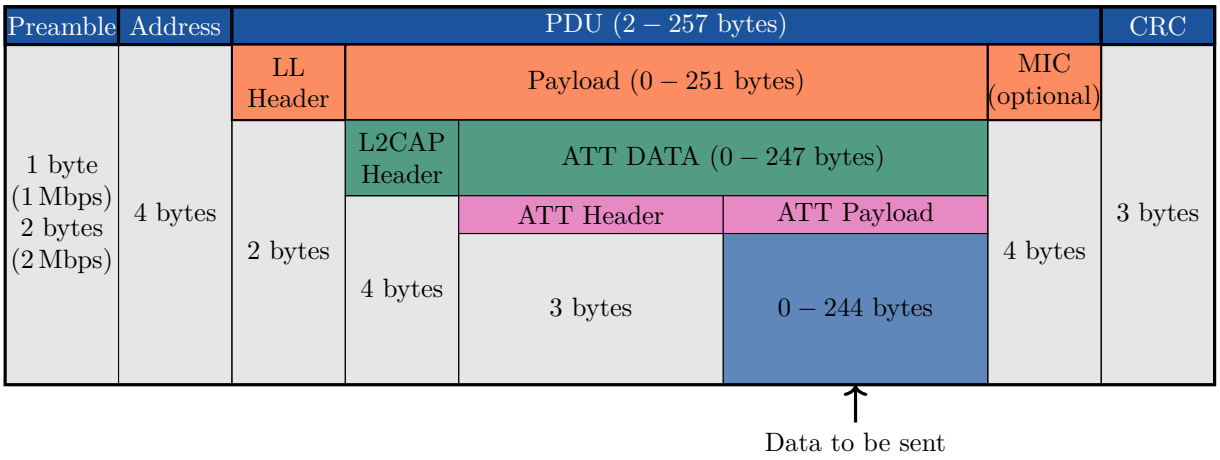


FIGURE 2.11: Detailed BLE v5 Link Layer packet structure.

It is essential to stress that a 2 Mbps bitrate does not mean that the data is exchanged at 2 Mbps. As the FIGURE 2.11 shows, the actual data, known as the ATT Payload, only forms a fraction of the LL packet. Depending on the size of the payload, the presence of the MIC (see later), and the bitrate, this fraction can vary from 0 (empty) to 92.08 %. All the additional content is not relevant for the end-user, and this information can be seen as an overhead to the packet. BLE traditionally transmits 1 bit per symbol (uncoded). Still, if one application requires an extended range, it is possible to encode 2 or 8 symbols per bit, thus reducing the data rate by a factor of 2 or 8.

On top of the packet content in itself, one must add the mandatory 150 µs Inter Frame Space (IFS) that should dictate the minimum time spacing between two consecutive packets. There are other limitations that, added altogether, make it unpractical to reach a 1 Mbps data rate when using a 1 Mbps bitrate.

FIGURE 2.12: Minimal spacing between two contiguous packets.

For example, [12] describes how the practical datarate of an optimal (minimum IFS – maximum ATT Payload) uncoded 1 Mbps is approximately 508 kbps.

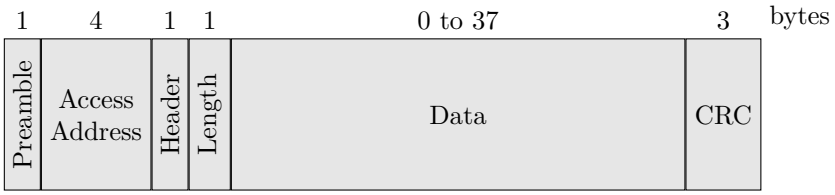


FIGURE 2.13: BLE pre-v5 Link Layer packet structure.

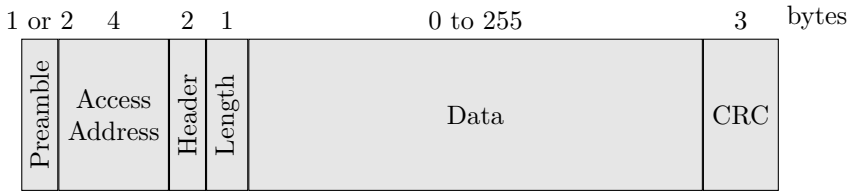


FIGURE 2.14: BLE v5 Link Layer packet structure.

2.3.3 Multiple communication modes

In opposition to Bluetooth Classic, BLE offers multiple communication modes:

1. one-to-one (also supported by Bluetooth Classic) for data transmission applications;
2. one-to-many (broadcasting) for location services applications;
3. and many-to-many (mesh) for network device applications.

Each mode uses its physical layer modulation and demodulation methods, so one must use the same mode for two devices to communicate. The one-to-one mode is currently the most used. However, broadcasting and mesh applications gained attention in the current BLE core specifications as the Bluetooth SIG Group continues to focus on developing these communication modes.

2.3.4 Advertising

As it was shown in FIGURE 2.3, there are two types of BLE channels: the advertising channels (37, 38 and 39) and data channels (others). The formers are exclusively used for pairing between devices and are placed carefully to avoid interference with Wi-Fi.

When two devices want to connect, they must undergo an advertising-scanning procedure. The peripheral advertises while the central scans the advertising channels to initiate a connection with the peripheral. BLE allows a wide range of parameters for those events:

- $T_{\text{AdvInterval}}$ can be any value between 20 ms and 10.24 s, in steps of 0.625 ms;
- T_{AdvDelay} is a pseudo-random value between 0 and 10 ms that helps to reduce the possibility of collision between advertisements of different devices;
- $0 < T_{\text{ScanWindow}} \leq T_{\text{ScanInterval}} \leq 10.24\,\text{s}$, also in steps of 0.625 ms, where the equality of the two parameters is called *continuous mode*.

A typical length for advertising packets is 128 µs, while it is 144 µs for data packets with a maximum of 376 µs [1, p. 31] for devices using BLE pre-v5. With BLE v5, the maximum packet duration goes up to 2.12 ms.

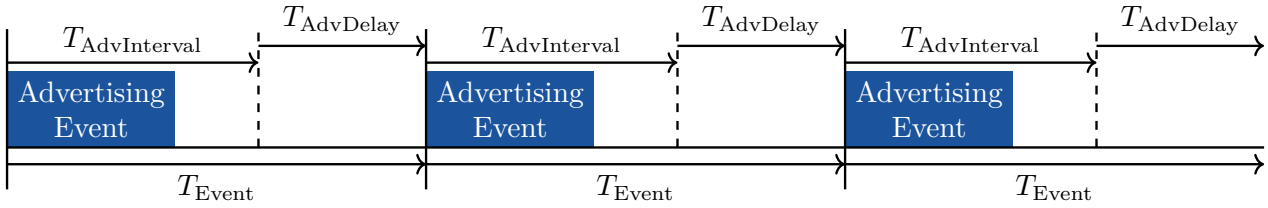


FIGURE 2.15: Avertising procedure [13].

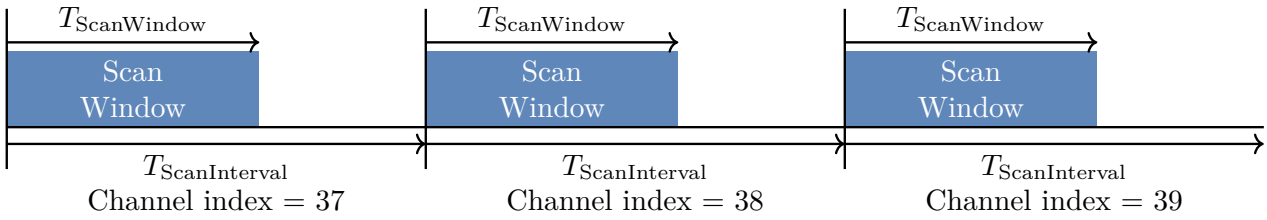


FIGURE 2.16: Scanning procedure [13].

Last but not least, the possibility to have very long advertising interval values allows to run on low footprint batteries for months, as it is shown in TABLE 2.2: the larger the interval, the longer the battery will last. This relationship is especially interesting for BLE devices placed outside urban areas, such as sensors used for forest environment monitoring [14]. They do not send a large amount of data but should have the most extended battery life possible. Here, some chipset configurations allow the device to last several years (4.76 years max.) on a battery (CR2477) that takes less than 4 cm^3 [15]. Even though the practical battery life will depend on many other parameters, such as the weather, the radio utilisation or the background processing tasks, TABLE 2.2 gives a good insight of the orders of magnitude that BLE devices can reach.

Chipset	Advertisement Interval	CR2032 small 240 mA h	CR2450 medium 620 mA h	CR2477 big 100 mA h
Gimbal	100 ms	1.0 month	2.5 month	4.0 month
Gimbal	645 ms	N/A	N/A	N/A
Gimbal	900 ms	N/A	N/A	N/A
Nordic	100 ms	1.1 months	2.8 months	4.5 months
Nordic	645 ms	5.4 months	13.9 months	22.4 months
Nordic	900 ms	11.0 months	28.7 months	46.3 months
Bluegiga	100 ms	0.6 months	1.6 months	2.6 months
Bluegiga	645 ms	5.8 months	15.1 months	24.3 months
Bluegiga	900 ms	13.9 months	35.8 months	57.1 months
TI CC254x	100 ms	0.7 months	1.8 months	2.9 months
TI CC254x	645 ms	4.3 months	11.2 months	18.0 months
TI CC254x	900 ms	5.6 months	14.3 months	23.1 months

TABLE 2.2: BLE chipset battery life, reproduced from [16].

2.3.5 Security

Nowadays, wireless communications are essential to most services. It is vital to ensure that the information exchanged between two endpoints can be kept secret to every other third party that is not part of the communication. BLE protocol does not derogate from this and provides the possibility to encrypt the information sent over the air.

The security of BLE communications is ensured by a 128 bit Advanced Encryption Standard (AES), in Cipher Block Chaining (CCM) mode, encryption of the packets. Without entering in too many details, CCM provides both authentication and confidentiality during data transfer. AES is one of the most popular encryption algorithms nowadays due to its robustness and relatively low power cost. For more details, one can find two very comprehensive and free tutorials about the encryption used in BLE on Texas Instruments and Nordic Semiconductor websites.

On top of this encryption scheme, each encrypted packet may also include a Message Integrity Check (MIC) value to authenticate the validity of a sender, as well as a packet counter to prevent replay attacks.

2.4 Robustness

One essential aspect of BLE is its robustness: it must work even in complex configurations with multiple other devices while providing a reliable transmission link. This robustness is achieved by leveraging several techniques, highlighting here the most relevant ones to this thesis.

2.4.1 Adaptive Frequency Hopping

As seen in Section 2.2, the ISM band is shared among many protocols, which can cause interference. In order to minimise this effect, many protocols use Frequency Hopping. This strategy consists of regularly changing the frequency on which data is exchanged to avoid staying in a lousy frequency with much interference or signal attenuation due to fading. For this purpose, the spectrum is divided into 37 different data channels (see Figure 2.17), and any one of these can be used for a connection.

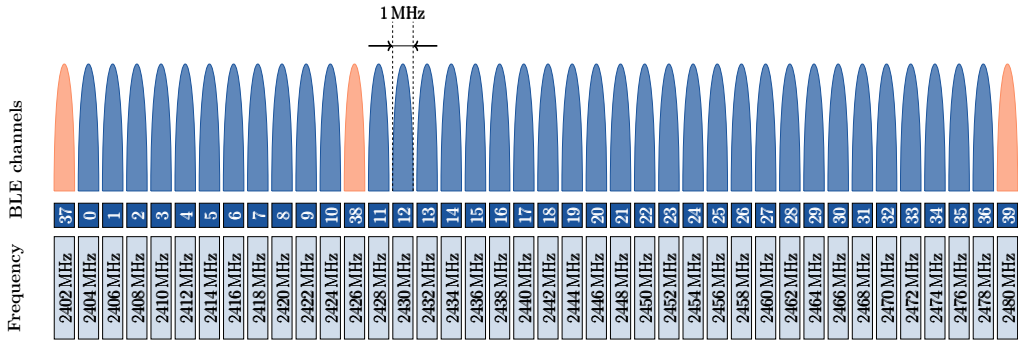
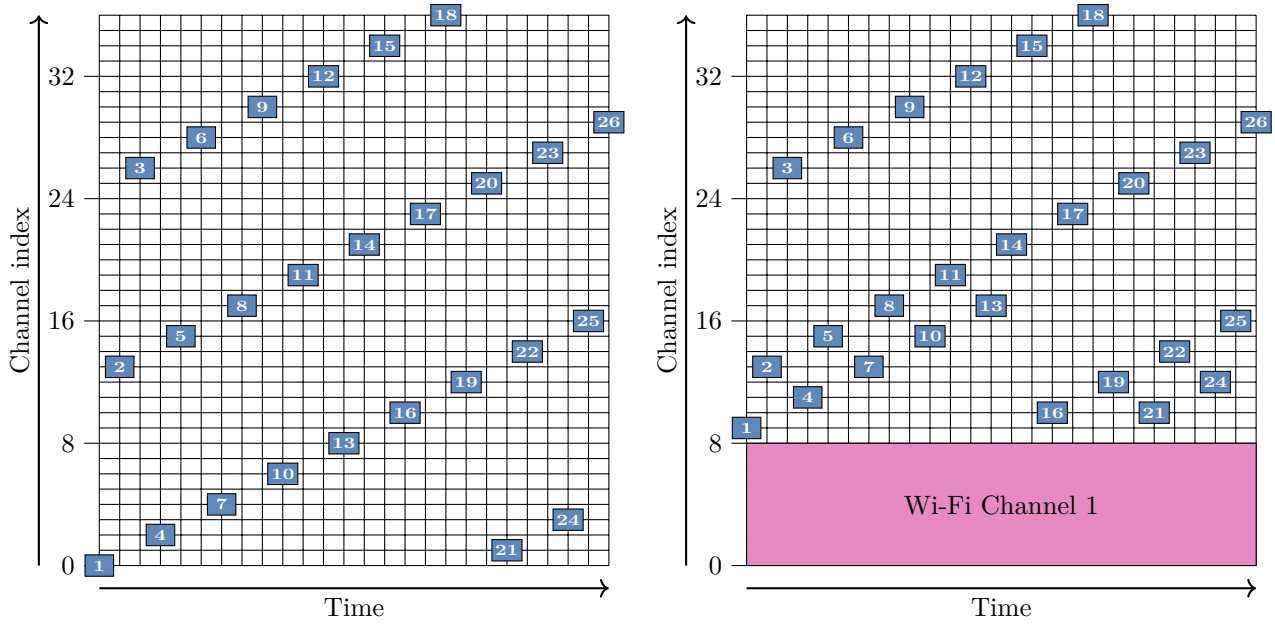


Figure 2.17: BLE channnels.

It is also challenging to intercept such a signal if the frequency-hopping pattern is unknown, making it more robust against potential radio jamming. Usually, the frequency used at a given time follows a pattern, called the *hop sequence*, that can be mathematically described as a number sequence using addition and modulo operands:

$$C_{i+1} \equiv C_i + A \pmod{37} \tag{2.1}$$

where C_i and C_{i+1} are current and next channel indices, A is the *hop value*, usually between 5 and 16 (included), and 37 is the number of data channels. An example of frequency hopping is detailed in Figure 2.18a: numbers in white depict the order at which the channels are selected, *i.e.*, the time index i, not the channel number.



(A) Frequency Hopping

(B) Adaptive Frequency Hopping

FIGURE 2.18: Example of frequency hopping and AFH with a *hop value* of 13.

The downside with the technique mentioned above is that channels are treated as being equally good. Nonetheless, as seen in SECTION 2.2, FIGURE 2.3, interference can occur on a very specific set of channels, such as data channels 0 to 8 if Wi-Fi channel 1 is active. In such situation, those channels should be removed from the list of good channels and not used in the sequence for better robustness.

To this end, the BLE protocol uses Adaptive Frequency Hopping (AFH). AFH is a technique used to efficiently cycle through all the channels (frequencies) while considering the knowledge of good and bad channels. Hence, AFH extends fundamental Frequency Hopping by remapping the wrong channels to the good ones, as shown in FIGURE 2.18b. This FIGURE uses the first algorithm to be proposed by the BLE Core specifications [17, Sec. 4.5.8.2]. The latter is very simple to implement but does not use the channels uniformly. For high power BLE devices, this can lead to problems with the *TX Power for Regulatory Compliance* [18] guidelines that devices have to follow. In order to use the good channels uniformly, a second algorithm based on a pseudo-random generator is proposed [17, Sec. 4.5.8.3].

It is essential to note that, even though the BLE core specifications require the use of AFH, it does not detail how one should implement the channel classification. The details of implementation are left to the device designer, and the distinction between good and bad channels is significantly linked to the role of channel assessment, see SECTION 2.5.4.

2.4.2 Error checking

One of the choices made when designing Bluetooth Low Energy was not to include any Error Correction Code (ECC). BLE is thought to be low power, and error correction can be power consuming. Here, retransmission is preferred over packet overhead induced by ECCs. Nonetheless, it does not mean that BLE does not avoid errors: it detects them and efficiently.

A 24 bit Cyclic Redundancy Check (CRC) is used to detect all 1, 2, 3, 4, 5, and all odd bit errors. FIGURE 2.19 recalls the structure of a Link Layer packet with the CRC at the end.

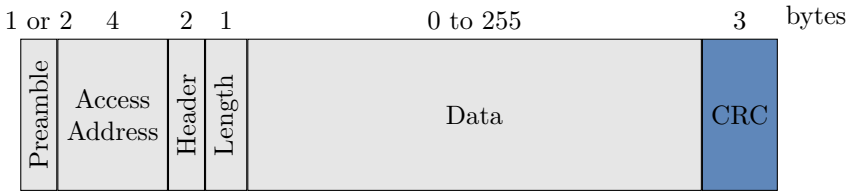


FIGURE 2.19: BLE v5 Link Layer packet structure with CRC part highlighted.

In the literature, attempts to use the CRC as an ECC have been done [19] but results were not outstanding, and one must first improve it before considering it an Error Correction Code.

2.4.3 Packet acknowledgement

Once in a while, data packets will not reach the receiver in the correct state. Those losses are often caused by the accumulation of interference and path loss, resulting in the deterioration of the transmitted signal. In such cases, it has been seen that the CRC can detect errors in the message. Optionally, the MIC can also check if the content of the message has been altered. Then, one very legitimate question may arise: *what happens when a packet is lost?*[1]

This problem could be solved if there was a way to confirm that a packet was received or not. Before introducing a solution, it is crucial to highlight that there are essentially two types of isochronous streams in BLE: Connected Isochronous Stream (CIS) and Broadcast Isochronous Stream (BIS). For example, both are relevant for audio transmission. However, they differ in how they operate: the former uses bilateral communications between the central and the peripheral, while the latter uses unilateral communications.

In the BIS case, there is no way to know that a packet was lost. The only option is to retransmit every packet multiple times to increase the probability that each packet was at least received once without any error.

Nevertheless, for CIS, a packet acknowledgement mechanism is used: every time the central sends a packet, it waits for an acknowledgement (ACK) from the peripheral before sending new packets. If no positive acknowledgement is received (NACK), then the packet is retransmitted. This scenario is depicted in FIGURES 2.20 and 2.21. If a subevent is unused, an empty packet that is 80 µs long will be sent instead.

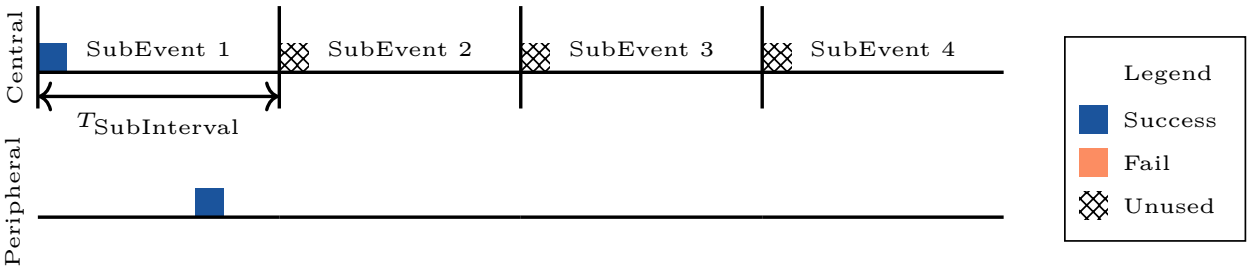


FIGURE 2.20: CIS - Acknowledgement of packets - No packet is lost.

[1] A packet is considered to be lost as soon as it contains errors.

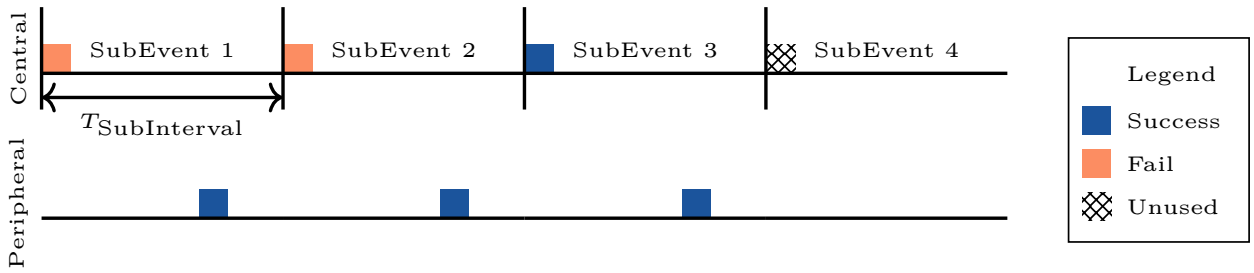


FIGURE 2.21: CIS - Acknowledgement of packets - Two packets are lost.

The maximum number of retransmitted packets is left to the designer. Increasing this number will lower the Packet Error Rate (PER) but increase the latency or bandwidth usage. It is important to note that subevents are only used when the acknowledgement from the peripheral indicates that the packet was lost. If no acknowledgement was received, the central assumes that the packet was lost and will retransmit the packet once more. If the packet was received, but the acknowledgement was lost, as depicted in FIGURE 2.22, then the connection enters in an undesirable state. The peripheral will not listen to the next subevents, and the central will use all retransmissions possible, thinking that the peripheral does not receive them. This scenario causes the device to consume more power than needed and shows one of the issues of this system. Nonetheless, this could be solved using a double acknowledgement mechanism, for example.

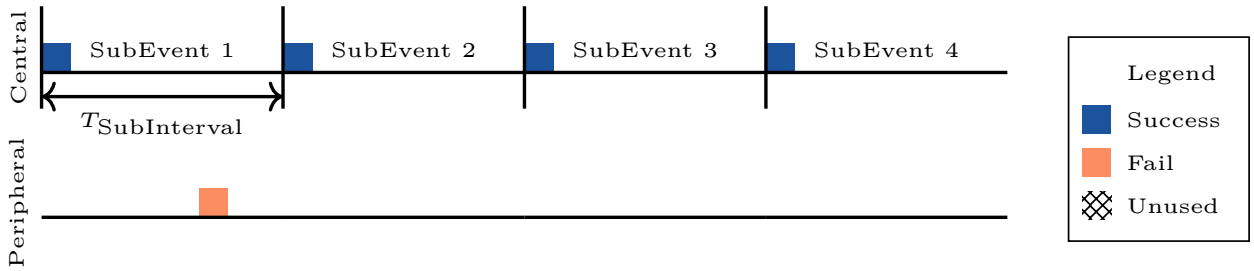


FIGURE 2.22: CIS - Acknowledgement of packets - Acknowledgement is lost.

One last critical aspect is: *how is the acknowledgement mechanism implemented?*

As actively using radio costs power, sending shorter messages helps to reduce the power consumption. Therefore, the Bluetooth SIG Group developed a technique to acknowledge packet with minimal radio on time. In the data header of each packet, one bit is dedicated to the Sequence Number (SN) and a second bit is dedicated to the Next Expected Sequence Number (NESN). When a device receives a packet, it compares the received SN with the previous NESN it sent. If they are different, it corresponds to an ACK: packet was received, waiting for a new packet (NESN is different). If both bits have the same value (all 0's or all 1's), then the packet was not correctly received: it is a NACK. Since both bits are contained in the data header, empty packets can be used for packet acknowledgement.

2.4.4 LE Power Control

As introduced earlier, the current v5.2 BLE specifications add the possibility to control the transmit power for efficient power management and reliable data transmission in various environments. Despite these advantages, power control is not always helpful and can increase the complexity of communications.

For the devices that are currently designed at NXP Semiconductors, LE Power Control is not needed. Devices run already on very low power specifications, with a current consumption somewhere between 3 mA and 7 mA, and stick to maximum power all the time to benefit from the best range possible. On top of that, the transmit power is only a tiny fraction, usually 10 % to 20 %, of the total power consumption. Reducing the power of radio communications does not have a significant impact here. Devices with larger power amplifiers,

such as smartphones, could benefit from power control since they can work on a more extensive range of transmit powers.

2.5 Scheduler

The scheduler is one of the critical components in BLE technologies. In every embedded system, a scheduler is needed. It must, as its name implies, decide which task should run at which time. Each application has its design constraints, and the scheduler should then be adapted to fit the specifications best. Some tasks run periodically, such as connections between devices, and other tasks can run aperiodically, such as the channel assessment, but should run as often as possible to provide robustness. The scheduler's role is also to make sure that no two tasks access the radio simultaneously. This individual access constraint is also valid if there are other pieces of hardware that the device can use: only one task can access the same hardware at a time. This constraint forces the scheduler to reject some tasks and set up a priority order for each of them.

BLE is popular because it stays robust while offering decent data rates and data ranges at low power. Among many, one of the critical tasks that the scheduler has to do well, and to do quickly, is the so-called channel assessment.

The following Sections emphasise the essential roles that the scheduler should fulfil.

2.5.1 Task selection

As described just above, the scheduler has to order the execution of the tasks. To this end, a priority scheme is often used to allow the scheduler to compare two tasks together and tell which one should be executed first. The choice of the schedule parameters is left to the designer: a solution working for wireless earphones may poorly impact IoT sensors. Many task priority schemes, such as Round-robin or Starvation mechanism, exist and offer different pros and cons. If the scheduler uses some dynamic priorities, as with the Starvation mechanism, it must also update the priorities accordingly.

Not only are the tasks selected based on their priority, but the selection also depends on the timing constraints of the tasks: it can be strict, *i.e.*, the task needs access to the radio at exact requested timing, or it can be the opposite, flexible. For example, tasks related to connections are typically fixed. Usually, those connection-oriented tasks are given a higher priority to better meet timing constraints.

Each task that expires, *i.e.*, that did not receive radio access in time, will be processed in the background.

2.5.2 Optimised timing

When a radio task is created, a new timing anchor must be allocated. Once it is done, all subsequent event timings will depend on this first timing anchor. Choosing an appropriate anchor for those tasks is very important as it will directly impact the number of future conflicts with other existing tasks.

To this end, the scheduler has to search and find an appropriate anchor before starting a new connection or before advertising to minimise the impact on the other tasks.

2.5.3 Queue managing

Since the tasks are mainly processed one by one[2], the scheduler can organise a queue with all the tasks (Figure 2.23). Once a task has completed, the queue will update to reflect the following tasks to be processed. This queue can either be ordered or unordered. In the first case, the scheduler may need to insert a task at a specific position in the queue to get it to run quicker.

Something that can also happen is called task pre-emption. Pre-emption can take place if a task with higher priority needs access to a resource that is locked by a lower priority task. In this case, the lower priority

[2]They could be processed in parallel if there were multiple cores.

task is required to stop and will continue its processing later. The scheduler must also handle these slow and hard handovers between tasks.

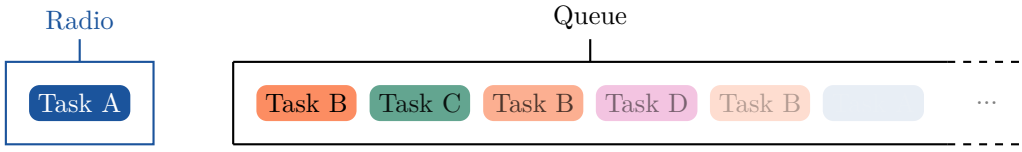


FIGURE 2.23: Tasks queuing up before accessing the radio.

2.5.4 Channel assessment

As seen in SECTION 2.2, the BLE channels are using similar frequencies as other prevalent protocols. SECTION 2.4.1 presented the AFH method, which relies on a *hop sequence*, as a way to minimise interference. The problem is: *how to find a good hop sequence?* Such a thing is possible only if a map describing the quality of each channel is known so that lousy channels are avoided. Evaluating the quality of channels is the role of the Channel Assessment (CA) task. The absence of any CA is often referred to as No Channel Assessment (NOCA).

This background task must be quick not to disturb higher priority tasks but must also be efficient. If the quality of the channel is not evaluated well enough, the *hop sequence* generated will not be good, and the PER will start to increase. This task is thus a pillar in the design and will be further studied in the following CHAPTERS.

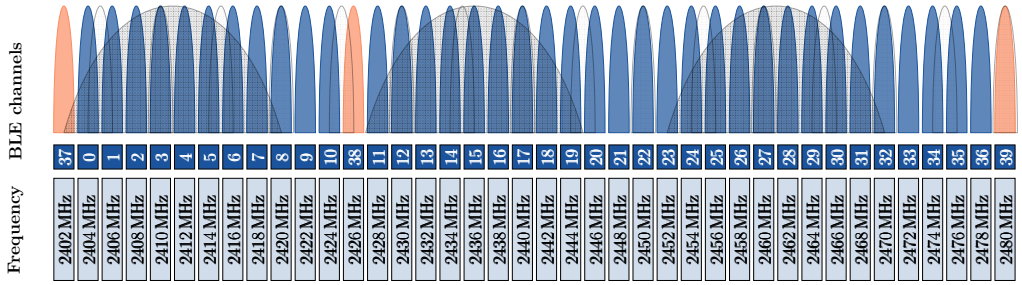


FIGURE 2.24: BLE channnels suffering from potential interference: Wi-Fi 802.11 and ZigBee channels are drawn on top with crosshatches.

FIGURE 2.24 highlights the impact of interference on BLE channels and why channels 37, 38 and 39 were chosen to be the advertising channels: they are located outside the Wi-Fi's three most used channels (1, 6 and 11). ZigBee technology is far less prevalent than Wi-Fi or Bluetooth Classic, so potential interference is less problematic.

It is also essential to keep in mind that BLE devices emit, on average, with much less power than their Wi-Fi counterparts, making them more vulnerable to interference.

The following CHAPTERS will introduce different algorithms and metrics to assess the channels and measurements to compare the theory and practice.

State of the art

3

The previous Chapter presented the main features and components of the BLE technology. The present Chapter will discuss the modern BLE applications and the different challenges that they leverage. This Chapter will also give some values and orders of magnitude for various quality indicators and parameters that the BLE devices rely upon.

3.1 Modern BLE devices

Historically, BLE devices were straightforward: a single connection was held at any time, making the task of scheduling straightforward, and devices density was not a concern. Assuming such simplicity is not valid anymore as IoT devices are present everywhere. Every single one can be connected to a smartphone, a tablet or some remote access third device. As those devices often run continuously in the background, one central device must handle multiple connections concurrently not to lose any bit of information that a specific device, *e.g.*, a temperature sensor, might return.

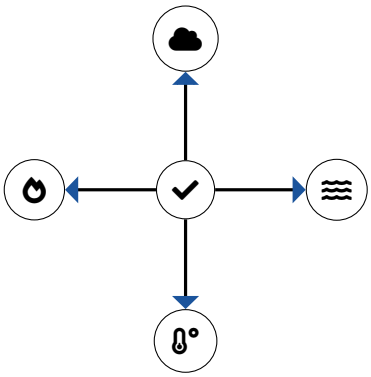
This challenge of multiple connections is not specific to IoT devices but has been gradually brought by the seemingly never-ending growth of connected devices. In the past, a smartphone could only connect to the cellular network for simple text messaging and call handling. Now, it is possible to live stream video from a camera on the Internet via the cellular network while being outside, and at the same time use some wireless earphones to listen to music. Once they have access to a Wi-Fi network, they may switch to turn the cellular network off and stream using Wi-Fi. For a good user experience, the handover between these two protocols should be as soft as possible and automated.

In BLE technologies, similar growth is observed. Many wireless earphones are running on tiny batteries, making the BLE protocol a good candidate for audio transmission. Users often experience high inter-devices connectivity, especially if they are from the same manufacturer or at least the same brand. Users expect to find the same seamless connectivity when using wireless earphones or portable speakers. The audio wireless link is not the only example of BLE multi-connectivity. However, it is one of the most spread out. It draws much attention given the interest that people have in its applications: wireless earphones, audio broadcasting in public areas, portable speakers, hearing aids, and all the others.

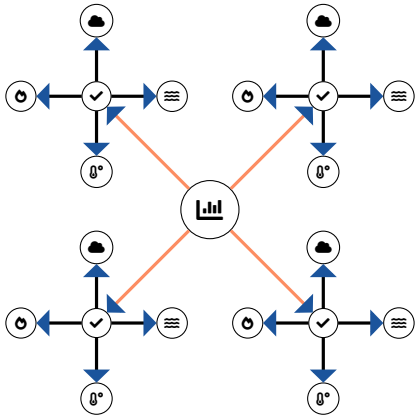
In the following Sections, two examples of state-of-the-art BLE use cases will be presented. The first example illustrates an example of mesh topology BLE network. It mainly serves as an introduction for future areas since the specific challenges leveraged by this kind of applications are not covered in this paper. The second example is more practical as it is closely related to what is currently developed at NXP Belgium. As such, the following Chapters will mainly focus on the challenges raised by this example.

3.1.1 Highly connected company

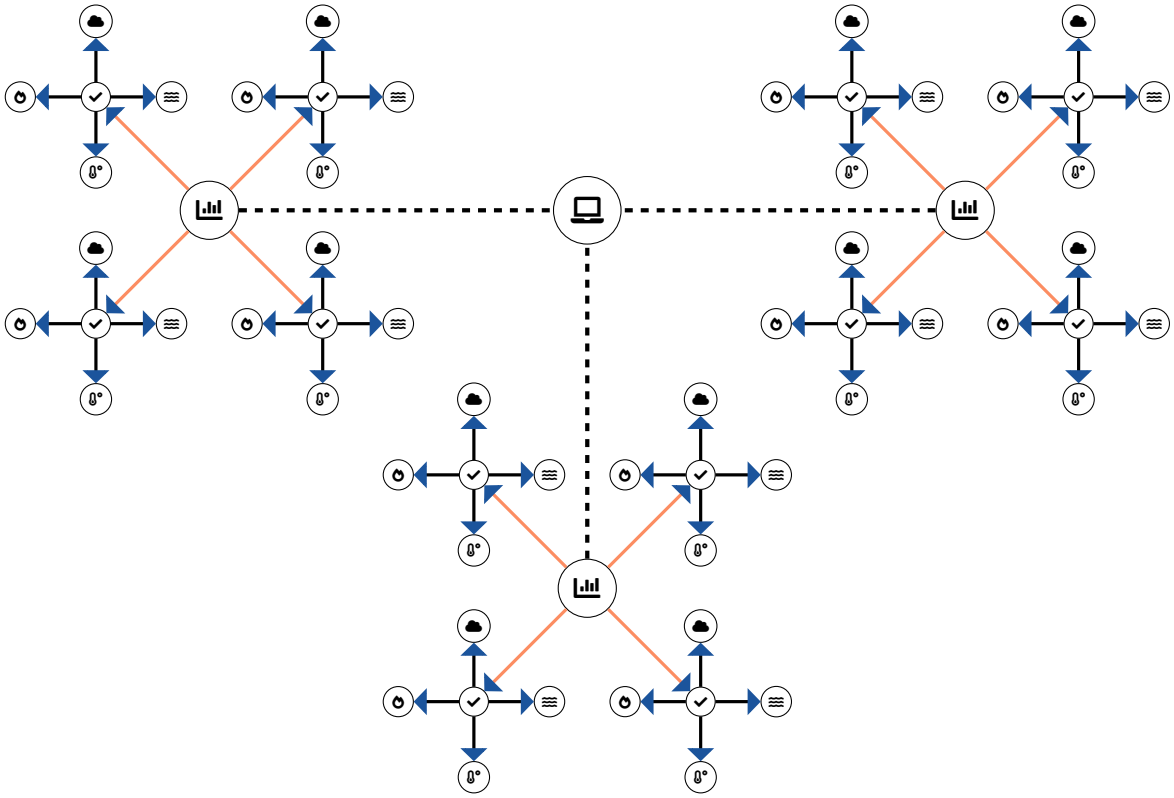
One can imagine the situation where a large company wants to monitor the temperature, the CO_2 level, the smoke and the humidity at every building level. To this end, the company splits each level into multiple areas where each of them contains a set of BLE sensors, one for each quantity to be measured. Each of those sites also includes one extra BLE device that is the central of the other four devices and checks that every value sensed is standard, FIGURE 3.1a. Each central is, in turn, connected via BLE to a central device controlling the entire floor, FIGURE 3.1b. Finally, those centrals are connected via cable to the main computer that can trigger alerts and regularly reports the values read by the sensors.



(A) Company BLE network per area



(B) Company BLE network per level



(C) Company BLE network

FIGURE 3.1: Example of a highly connected company BLE network with 4 areas per level, and 3 levels.

This very hierarchical organisation simplifies the relationship between devices and their role: the computer at the top of the chain controls every single storey, which, in turn, manages every local area. In short, every device is, at most, the peripheral of one connection and knows that a conflict between two or more centrals will never happen. One downside with this approach is that the number of simultaneous communications can rapidly become large, and the analysis of such mesh topology can become quite complex. This applications leverage a set new paradigms, but are covered here.

3.1.2 Hearing aids in home environment

People suffering from hearing problems, such as deaf or older people, can benefit from hearing aids to easily converse with other people, hear when the doorbell is ringing, listen to music or watch movies on the TV. As carrying cables all around people's head is not convenient, those hearing aids are primarily wireless. Again, the power consumption aspect of these devices is essential, and BLE is an excellent way to provide wireless communication at low cost and low power.

With those applications being introduced, there is a wide variety of devices to which hearing aids can be connected, and the end-user may want to be connected to several of them simultaneously. Indeed, while watching a movie using a wireless audio link, one person may receive a call from his smartphone or hear that someone is ringing the doorbell. It is crucial to constantly keep contact with all those devices to hand over to them once they require attention quickly.

This scenario gets even more complex if earphones are separated into two, left and right, devices. Then, a connection must be held between the two devices to keep the audio synchronised. Additionally, other devices can also have a BLE connection with those products to allow seamless handover between multiple devices. A very realistic situation is depicted in FIGURE 3.2, and one of the goals of this thesis is to provide the necessary material to appreciate the complexity of those modern applications.

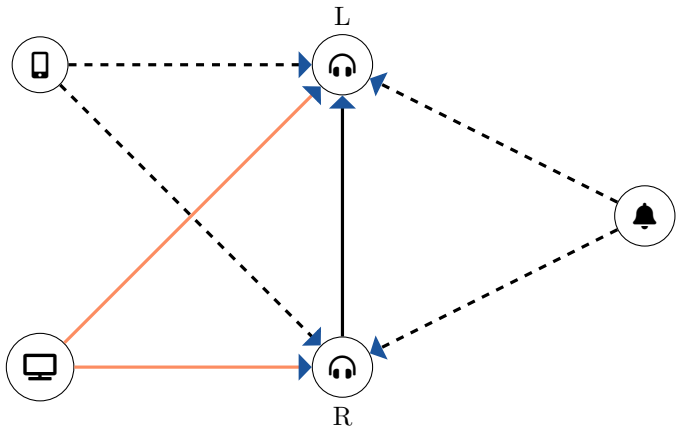


FIGURE 3.2: BLE Hearing aids in home environment. Connections in orange are active audio streams and dashed ones are audio connections awaiting to be active.

Compared to the previous scenario, this one presents one significant difference: some devices are the peripheral of multiple BLE connections. Having numerous links means that the scheduling solution may not be as simple as before, and conflicts can occur between the different centrals of the same peripheral. SECTION 3.2 details the main challenges of those modern applications.

3.1.3 NXP's BLE products

In this SECTION, some of the latest products developed by NXP Leuven will be briefly presented. Their personal health centre segments its BLE market in three different areas:

1. **Hearing instruments:** aiming low power digital signal processing and wireless audio streaming. Those

technologies are to be embedded into hearing aids and implants;

2. **Wireless personal audio:** low power BLE audio streaming for gaming headsets, speakers and earbuds;

3. **Health IoT:** connectivity chips with sensor frontend for condition monitoring and therapy support.

As described above, NXP's products are mainly targeting devices in close contact with humans. Such products are often required to meet high-end standards as a lack of tolerance in their design could dramatically impact people's health. This constraint showcases the necessity of designing robust products and understanding the challenges that engineers must face.

In addition to what is mentioned above, NXP also has (at least) two other groups working on BLE technologies. The first branch focuses on more general-purpose products (not only health), and the second is developing Wi-Fi and BLE combo devices, *i.e.*, devices working with both technologies.

3.1.4 Typical BLE connections

Finally, to wrap up with the modern applications of Bluetooth Low Energy, Tables 3.1 and 3.2 summarise common types of ACL and audio connections that are present in BLE devices, especially those developed by NXP Leuven. Those values will be used to properly design the algorithms and techniques discussed in the next Sections and Chapters. In Table 3.2, the bandwidth refers to the ratio of time taken by one event over the interval duration. As earlier described, one event can be split into multiple subevents to improve, for example, reliability by retransmitting multiple times the same packet.

Link description	Interval (ms)	Average data content
GenericACL, suitable for most CIS connections	60	Idle most of the time, with 90% empty packets and 10% short control packets (20 B to 30 B) in both directions
FastACL	7.5	Usually intended for low latency control / gaming / keyboard. In active direction, 50% packets are 30 B long and 50% are 10 B long, and empty packets in the other direction
FastACL for quick audio setup	10	Intensive data packets and multiple exchanges per event during setup, but limited in duration. About 30 packet exchanges and then idle
HID_ACL	11.25	Mouse / keyboard. In active direction, 50% packets are 30 B long and 50% are 10 B long, and empty packets in the other direction
Responsive ACL	30	Idle most of the time, with 90% of empty packets and 10% of short control packets (20 B to 30 B) in both directions
Regular sensor ACL	100	Idle most of the time, with 50% empty packets and 50% short control packets (20 B to 30 B) in one direction
Idle ACL	250	Idle most of the time, with 90% empty packets and 10% short control packets (20 B to 30 B) in one direction
Slow ACL	100	Idle most of the time, with 90% empty packets and 10% short control packets (20 B to 30 B) in one direction
Relative prime control connection	62.5	Idle most of the time, with 90% empty packets and 10% short control packets (20 B to 30 B) in one direction
High data throughput ACL	25 to 50	Assuming bulk transfer, like firmware images, stream data. Using packet length extension sending 128 B packets at an average rate of 64 kbitps. Results in 1 to 3 ACL packets per event

Table 3.1: Common BLE ACL connections, provided by Sam Geeraerts.

Link description	Interval (ms)	Payload size (B)	Number of subevents	Bandwidth usage (%)
Low latency – CIS	7.5 to 10	25 to 160	3	25 to 50
High reliability – CIS	7.5 to 10	25 to 160	4 to 9	50 to 75
Low latency – BIS	7.5 to 10	25 to 160	3 to 5	15 to 75
High reliability – CIS	7.5 to 10	25 to 160	5 to 10	25 to 80

TABLE 3.2: Common BLE audio connections, provided by Sam Geeraerts.

3.2 CHALLENGES

As presented in the previous SECTION, modern BLE devices must handle very complex scenarios involving many challenges that are inherently linked altogether. This SECTION introduces some of these challenges and provides an analysis of how they can be addressed.

3.2.1 Packet collisions

Today's BLE devices can concurrently hold multiple connections, which leverages some severe challenges about the scheduling. If all the communications are directed from the same central device, everything can be predictable since each peripheral will follow directives from one unique device. However, as soon as there are multiple centrals in interleaving connections, this creates uncertainty about what can happen.

One can highlight two significant sources of randomness in connections: the clock drift and the packet error rate, and they will be discussed after the problem of packet collisions has been detailed.

Analogy to car collisions

An analogy between packet collision and car collision can be done and is presented in FIGURE 3.3. The underneath objective is to determine at which rate and when two cars may collide. To this end, one can imagine that the two vehicles are driving at a known speed V. The initial distance between the vehicles, d, is also known.

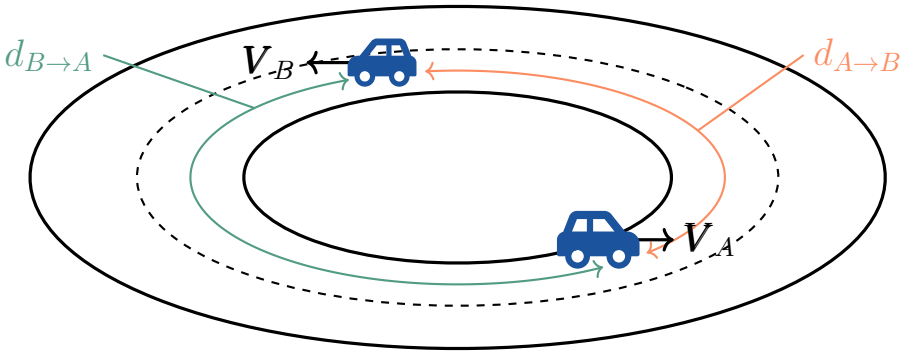


FIGURE 3.3: Illustration of the collision between packets. Cars (packets) are driving in a circle, in a never-ending loop (connection with periodic events). Each car has its speed (frequency), and the difference in their speed will cause the distance between the two cars to change over time.

Then, the time until first collision of the two cars, t_c^1 is:

$$t_c^1 = \begin{cases} \frac{d_{A\to B}}{V_A - V_B} & \text{if } V_A - V_B > 0 \\ \frac{d_{B\to A}}{V_B - V_A} & \text{if } V_A - V_B < 0 \\ \text{never}^1 & \text{otherwise} \end{cases} \tag{3.1}$$

If cars A and B have, respectively, length L_A and L_B, then the collision duration, *i.e.*, the time during which the cars have parts that can touch each other vehicle, is:

$$t_{\Delta c} = \frac{L_A + L_B}{|V_A - V_B|} \tag{3.2}$$

This value can also be seen as the convolution of two rectangular pulses, each having its respective car length (see FIGURE 3.4).

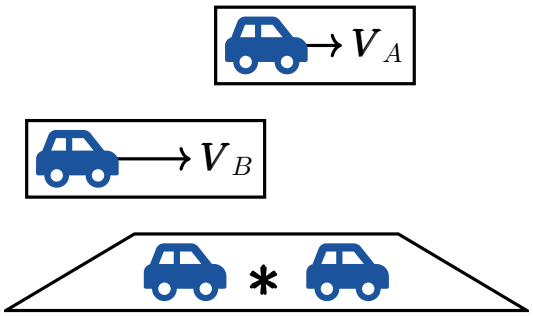


FIGURE 3.4: Illustration of the time window convolution using car lengths as window lengths.

Then, the time until second and all other subsequent collisions is:

$$t_c^{2+} = \frac{d_{A\to B} + d_{B\to A}}{|V_A - V_B|} \tag{3.3}$$

Finally, the period between two collisions, T_c, and the relative time spent in a collision, $T_{\%c}$, can be expressed as:

$$T_c = \frac{d_{A\to B} + d_{B\to A} + L_A + L_B}{|V_A - V_B|} \tag{3.4}$$

$$T_{\%c} = \frac{L_A + L_B}{d_{A\to B} + d_{B\to A} + L_A + L_B} \tag{3.5}$$

Application to communication packets

For packet collision, one can apply the same reasoning to obtain collision timings. Every connection represents a car, and every packet is an image of the vehicle at a given time instant.

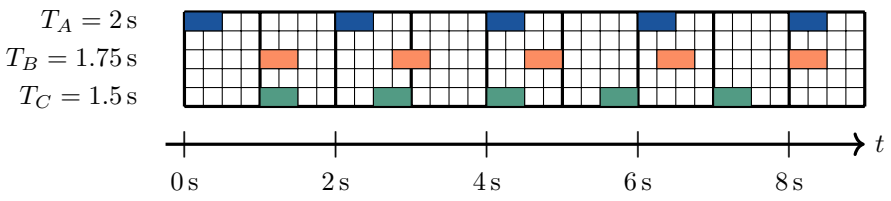


FIGURE 3.5: Example of a set of 3 connections with different period durations.

[1] Or always if they start in collision.

Here, the speed of a packet is actually its period: it may be counter-intuitive at first look, but taking the frequency would not give correct results when applied to the same problem as described with cars (FIGURE 3.2.1). To be more specific, the speed of any given packet is in $\mathrm{s\,period^{-1}}$ (second per period).

With the example of values in FIGURE 3.5, one can obtain:

$$t^1_{c,AB} = \frac{\frac{1}{2}}{2-\frac{7}{4}} = 2\,\mathrm{period}^{-1} \qquad t^1_{c,AC} = \frac{\frac{1}{2}}{2-\frac{6}{4}} = 1\,\mathrm{period}^{-1} \tag{3.6}$$

The period^{-1} is crucial since the result must be multiplied by a reference period duration to obtain a time in seconds. Therefore, the result will depend on the reference frame that is used: from connection A, the first collision with B takes place after $2 \cdot 2 = 4\,\mathrm{s}$, whereas, from B, this value is $2 \cdot 1.75 = 3.5\,\mathrm{s}$. The same reasoning applies to collisions between A and C, or B and C.

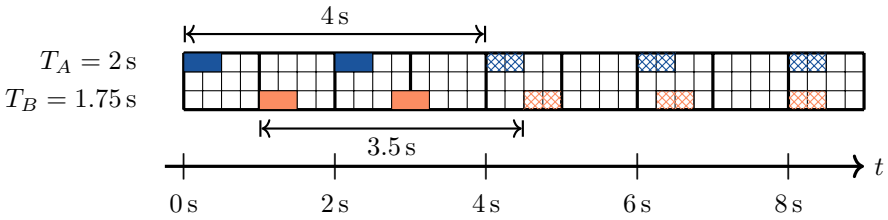


FIGURE 3.6: Example of collisions between two connections, where colliding events are filled with crosshatches.

Thanks of the discrete nature of telecommunications, each collision is likely to happen only the next time(s) a packet is sent, after time t^1_c. Indeed, connections A and B do not overlap after 4 s, but only start to collide starting on the subsequent events. The time during which packets will collide can be obtained by calculating $t_{\Delta c}$:

$$t_{\Delta c,AB} = \frac{1}{2-\frac{7}{4}} = 4\,\mathrm{period}^{-1} \qquad t_{\Delta c,AC} = \frac{1}{2-\frac{6}{4}} = 2\,\mathrm{period}^{-1} \tag{3.7}$$

Again, these values must be multiplied by a reference period to obtain a time duration in seconds.

Finally, packet collisions can be **deterministic**. When two or more connections are deliberately using different periods. For example, when one knows that it is impossible to fit all communications during the same period, different period lengths are taken to force some collisions, but with the largest period possible. Packet collisions can also be **stochastic** due to imprecision in the crystal oscillator frequency, the topic of next SECTION.

3.2.2 Clock accuracy

Clocks are key components in modern technologies since they give rhythm to almost every task performed. Their accuracy is therefore crucial, and one can measure their quality via multiple metrics. In short, a clock frequency can be represented as an ideal – or target – frequency, f_N, plus some frequency drift, Δf, see (3.8). This drift can take both positive and negative values and is not necessarily constant.

$$f = f_N + \Delta f \tag{3.8}$$

In itself, the frequency drift, Δf, is a function of multiple parameters [20, 21]: time, temperature, ageing, crystal's internals, etc. In most crystal oscillator datasheets, the quality of a crystal oscillator is described by three parameters: *(i)* tolerance, *(ii)* stability over temperature, and *(iii)* ageing. Typical values are given in parts per million (ppm) (ppm/year for ageing) which describes a maximum relative drift from ideal frequency. The tolerance error, noted here ρ, is linked to randomness in frequency distribution during the manufacturing process and results in a frequency which can be, at most, off by ρf_N Hz, at room temperature. Since the stability over temperature depends on the temperature range that a given crystal is capable of, it is often more

convenient to work with the temperature coefficient [22], in ppm/°C^2, where the reference temperature is, in most cases, 25 °C.

In addition to clock drift, the clock jitter (see FIGURE 3.7) will also increase the uncertainty about the timing and, therefore, require the radio to listen longer to compensate. For example, one can take a clock with an accuracy of 1000 ppm and a maximum jitter of 16 µs. An event which happens 1 s after synchronisation will require the device to start listening at least 1016 µs[2] earlier and continue to listen 1016 µs after, *i.e.*, more than it would have needed in perfect conditions.

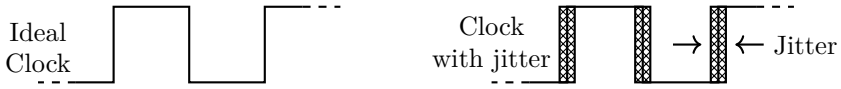


FIGURE 3.7: Illustration of the clock jitter.

Active and sleep clocks

As mentioned in the BLE v5.2 core specifications [17, Page 2928], BLE devices use two kinds of clock accuracies: an active clock accuracy and a passive clock accuracy.

The active clock is used for packet transmission during a connection, BIG or CIG event, active scanning, and when requesting a connection. This clock should have a maximum drift of ±50 ppm and less than 2 µs of clock jitter. Its frequency is often around 32 MHz.

On the other hand, the sleep clock is used for everything else and should have a drift equal or less than ±500 ppm, with a maximum jitter of 16 µs. In comparison with the active clock, the sleep clock runs at a much lower frequency, often around 32 kHz.

Quality	Tolerance (ppm)	Temperature coefficient (ppm/°C^2)	Ageing (ppm/year)
Good	±20	−0.04	±5
Very good	±10	−0.03	±3
High	±5	−0.03	±3

TABLE 3.3: Typical values of crystal oscillator quality indicators, for 32.768 kHz clocks found on Farnell.

For example, using a 500 ppm crystal oscillator, a fresh-new 32 MHz clock could have its actual frequency, at room temperature, ranging from 31.9984 MHz to 32.0016 MHz, meaning it can be off by almost 1.6 kHz from ideal frequency.

One interesting scenario is to translate this frequency drift into a period drift:

$$f = f_N + \Delta f \iff T = T_N + \Delta T \tag{3.9}$$

$$T \triangleq \frac{1}{f} = \frac{1}{f_N + \Delta f} = \frac{1}{f_N} - \frac{\Delta f}{f_N\,(f_N + \Delta f)} \tag{3.10}$$

$$\implies \Delta T = -\frac{\Delta f}{f_N\,(f_N + \Delta f)} \tag{3.11}$$

If Δf is proportional to f_N, such that $\Delta f = \rho f_N$, then (3.11) simplifies to:

$$\Delta T = \frac{-\rho}{f_N\,(1 + \rho)} \tag{3.12}$$

[2] $16\,\mu s + 1000\,\text{ppm} \cdot 1\,s = 16\,\mu s + 1000\,\mu s = 1016\,\mu s$

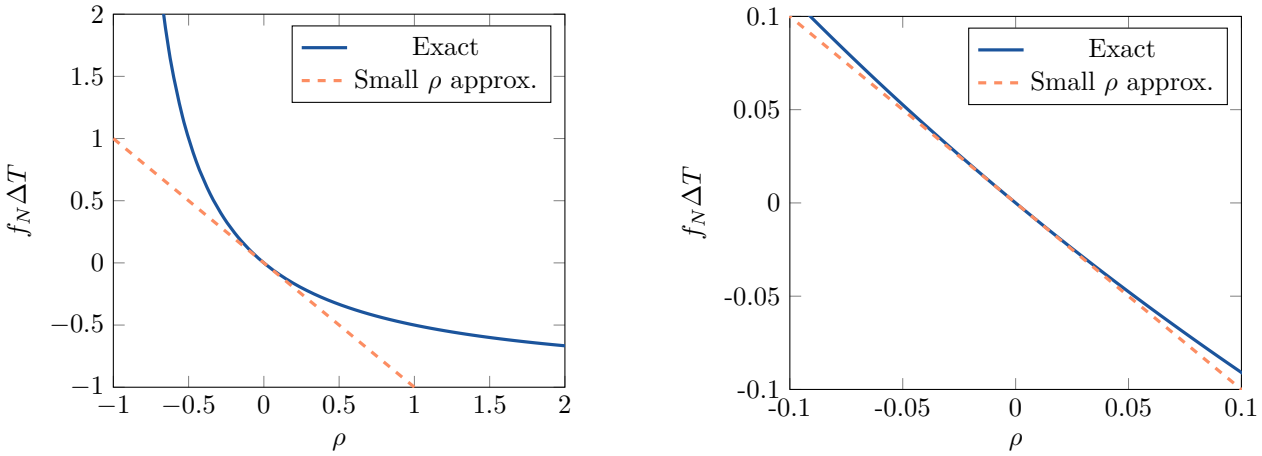


FIGURE 3.8: Influence of the relative drift ρ on the $f_N \Delta T$ product.

From FIGURE 3.8, one can observe two extreme cases: first, $\rho = -1$ implies an infinite period duration, since the frequency becomes zero. Second, $\rho = 1$ will divide the period duration by two. Nonetheless, such outliers are never encountered given that $|\rho|$ mostly lies below a few hundreds ppm, *i.e.*, $|\rho| \ll 1$. This means that (3.12) further simplifies to:

$$\Delta T \approx \frac{-\rho}{f_N} \tag{3.13}$$

The relative error, ϵ_r, done with this approximation is always below 0.05 % for values of ρ less or equal to ± 500 ppm, which will always be met as defined by the BLE specifications [17]. Finally, the relative error done by (3.13) is exactly equal to $-\rho$ since:

$$\epsilon_r = \frac{\frac{-\rho}{1+\rho} - (-\rho)}{\frac{-\rho}{1+\rho}} = \frac{\frac{\rho^2}{1+\rho}}{\frac{-\rho}{1+\rho}} = -\rho \tag{3.14}$$

Modelling Δf

As said above, the clock drift, Δf, or the frequency error, results from multiple factors. From measurements on real clocks [23, 24], one could legitimately want to model the frequency error as a random normal variable: $\Delta f_N \sim \mathcal{N}\left(\mu, \sigma^2\right)$. The mean, μ, is, ideally, close to zero, but what about the standard deviation σ?

Here, considering that ρf_N is the maximum tolerable clock drift, σ will be arbitrarily fixed such that $3\sigma = \rho f_N$. This uses the 3-sigma rule which assumes that 99.7 % of all values lie between $\mu - 3\sigma$ and $\mu + 3\sigma$.

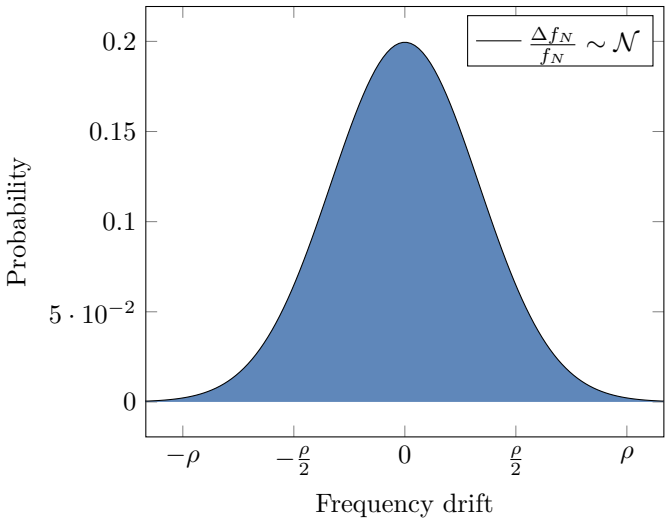


FIGURE 3.9: Drift's probability mass function in the frequency domain.

This simplified model assumes that the frequency is fixed. This simplification, therefore, eliminates frequency variation due to ageing or temperature changes. As it is often a fraction of the total offset, the frequency noise is not considered here. Throughout the same connection, the ageing effect is very negligible given that devices studied during this thesis are primarily used for connections lasting a few hours maximum, not months or even years. On the other hand, the temperature effect depends on the environment where the BLE device is used. However, the temperature effect will be considered negligible against tolerance error as devices of interest are not used in extreme weather conditions.

Here, the only drift that one should care about is the frequency drift throughout a given connection. Since synchronisation can be used at the beginning of a connection, the absolute drift from the nominal frequency is not as important.

3.2.3 Packet Error Rate

One crucial aspect in wireless communications is the notion of error: information is sent through a medium that can distort the original signal and introduce some errors. To model this random process, one can define that each bit of some signal has a probability p of success. The complementary of this value, $1 - p$, is known as the Bit Error Rate (BER).

Since a packet is made of multiple bits, a common hypothesis is to model all the bits as Independent and Identically Distributed (IID) Random Variables (RV), to obtain the following estimation for the error probability of one packet containing n bits:

$$\text{PER} = 1 - (1 - \text{BER})^n \tag{3.15}$$

For small values of BER, an excellent linear approximation can be made:

$$\text{PER} \approx n\text{BER} \tag{3.16}$$

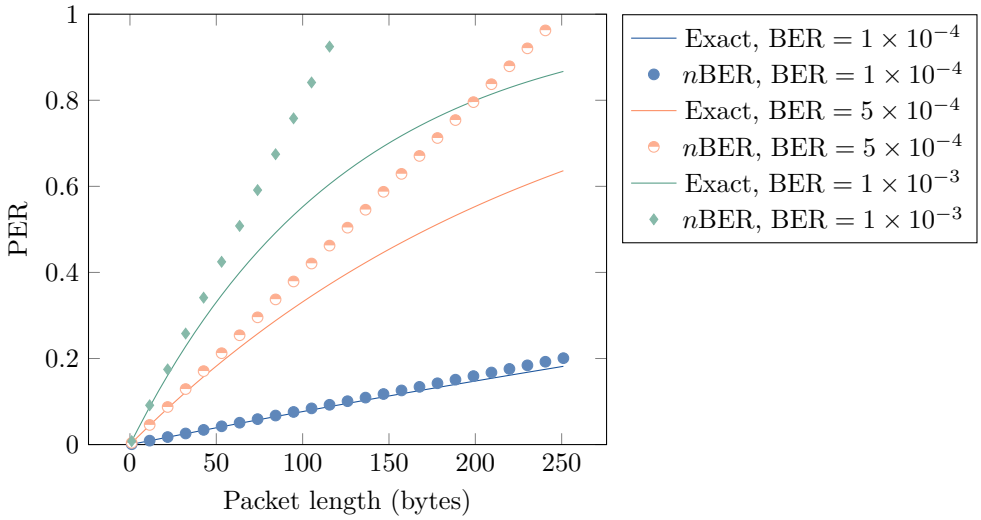


FIGURE 3.10: PER and its linear approximation as a function of the packet length, for various BER values. The range stops at 251 B, the maximum payload size.

For real applications, the PER is often below 0.1, hence roughly on the blue line or below, and the approximation (3.16) holds entirely accurate results with a relative error of 5 % or less. Hence, this approximation is used as a rule of thumb to quickly estimate the impact of various parameters on the PER and verify if it satisfies the device requirements.

While this assumption of independence is very convenient, one must stress that it may sometimes be erroneous: if some bits were lost due to interference, likely, subsequent bits in the same packet would also be lost. This dependence between errors is the reason why channels may be listed as bad if they experience a continuous deterioration over time, see the description of the channel assessment algorithm for more details (SECTION 4.3).

3.2.4 From a continuous problem to a discrete one

In previous SECTIONS, various challenges were addressed, assuming that the problems could be solved using a continuous approach. The problem is that the world of communications is intrinsically discrete: packets are sent periodically, and their length cannot take any value but a multiple of some base number. While this does not pose a problem for the study of clock drifting and PER, it impacts a lot the packet collisions as their connections are not continuously drifting toward each other but instead drifting by some constant amount at each period.

The latter means that the number of collisions over time also depends on other parameters, such as the connection timing anchor. SECTION 4.1, which describes the development of the slot finder algorithm, addresses the problem of computing the number of collisions in a discrete space.

3.2.5 Channel assessment

Regarding all the challenges previously discussed, they are all related to what is called the channel. As a reminder, BLE possesses 37 different data channels from which it can choose the one to use as a propagation medium for the information. A lousy channel could introduce bit errors in the packet due to interference between multiple connections. Therefore, it is critical to detect such channels and remove them from the list of used ones as quickly as possible.

A popular implementation of channel assessment is the Clear Channel Assessment (CCA). Primarily used in Wi-Fi and ZigBee's technologies for interference avoidance, CCA could still be used in BLE for interference detection. CCA listens to the BLE channels, trying to detect the preamble of Wi-Fi packets [25], for example. If a packet is detected, then it assumes another device occupies the channel. This technique mostly relies on

the assumption that one device keeps using the same channels: this is relevant for Wi-Fi connections but not for BLE ones. Trying to detect the channels used by another BLE device is pointless as it will constantly change over time due to Frequency Hopping. [26] compares different methods to identify channel utilisation and shows that a combination of RSSI measure and CCA (see Figure 3.11) can lead to above-90 % accuracy in interference identification. RSSI alone is also a good metric from which it is possible to reach 80 % accuracy or more. The advantage of the RSSI measure is that it is technology independent. Indeed, measuring the RSSI level does not need to know the packet structure and can detect BLE, Wi-Fi, and other protocols. Therefore, if only channel interference avoidance is needed, the RSSI can provide a good starting point.

On the other hand, if one needs to identify the technology that interferes within a given channel, it needs CCA. This method requires the device to implement the appropriate packet structure recognition for each technology it wants to identify. Technology identification would be helpful if one wants to avoid Wi-Fi interference without detecting other BLE devices.

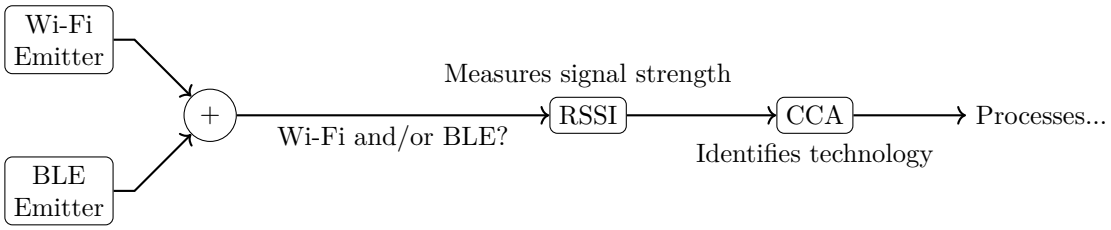


Figure 3.11: Channel assessment: how to identify the sources of interference.

In this thesis, the CCA is not used, as channel identification is not the primary concern. Indeed, the main source of interference in many scenarios comes from the domestic Wi-Fi router, and the trade-offs required to add CCA are not worth the cost.

Nonetheless, measuring the RSSI levels and choosing appropriate channels from them is already a challenging task. In an average scenario, a BLE device can only scan for 1 ms every 10 ms, *i.e.*, 10 % of the time. Within this 1 ms scan window, one must account for from 100 µs to 200 µs of overhead taken by the data processing and the radio acquisition. Therefore, only 800 µs to 900 µs is left for the actual RSSI measurement. If one wants to measure the RSSI level of two BLE channels during this duration, the overhead doubles, and the actual time spent measuring each channel drops down between 300 µs and 400 µs. As a result, it is more convenient to measure the RSSI level of one channel per scan window.

But measuring the RSSI level at a given time instant does not guarantee to detect anything. Indeed, it could be well that another device is using the channel but not emitting during the scan window (see Figure 3.12). This probability of detecting a Wi-Fi (or other technologies) signal, knowing that it is active, must be considered when designing the CA algorithm. Measurements in Section 5.3 try to estimate this probability in various situations.

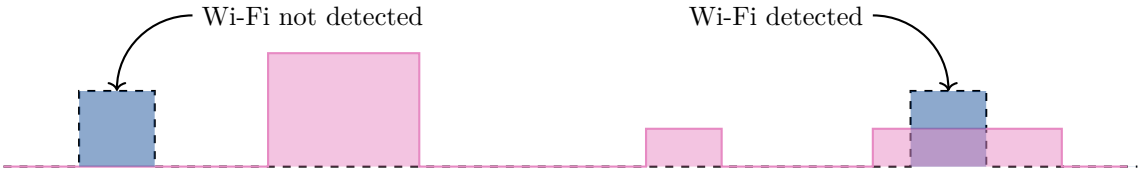


Figure 3.12: RSSI scanning in blue tries to detect Wi-Fi signal in rose.

Finally, using the RSSI measure may not be self-sufficient, and one may prefer to combine it with other metrics to enhance the algorithm's performances. As such, an algorithm using RSSI and PER measures is presented in Section 4.3.

Algorithms and metrics

4

Here below are discussed several algorithms and metrics that have been developed for the purpose of this thesis.

One must know that those were designed to work on very low power processors, such as the Cortex-M0 from Arm. This hardware limits the kind of operations that it can process: no floating-point operations, preferably no division, low memory footprint, and other constraints. Taking these constraints into consideration requires sometimes to use programming tricks and can become quite challenging. Lastly, for performance purpose, the algorithms are written in embedded `C`, and `Python` was only used for validation and testing.

4.1 Slot finder

Each connection must be allocated a timing anchor to determine when to start. The link is assumed to be periodic with a known period and maximum – or average – slot length from this first timing. The slot length is the sum of all packets that are sent without interruption. For a very basic use-case, this length is the sum of the packet length sent by the central, the IFS, and the empty packet (ACK) sent by the peripheral (see Figure 4.1). This duration can, of course, be more significant if more data needs to be sent in the course of one event.

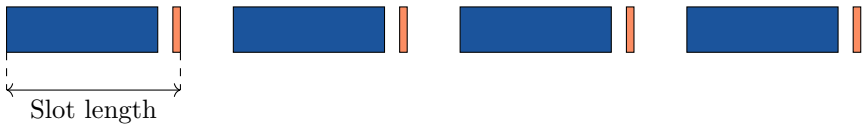


Figure 4.1: Slot length (or slot duration) principle.

As seen in Section 3.2.1, some timing configurations can produce collisions between data packets. The role of the slot finder algorithm is to find the best timing anchor for a new connection, given a set of already settled connections, that minimises the number of future collisions. One may also give more importance to some connections via the use of weights, *e.g.*, using the priority of each connection as its weight. Figure 4.2 shows an example of a connections set.

4.1.1 Problem definition

Before describing how the algorithm works, it is essential to define the problem and a cost function C that will depict how well the algorithm performs.

Let $\vec{t}^* = \begin{bmatrix} t_0^* & \dots & t_{n-1}^* \end{bmatrix}$ be the set of timing anchors of n already existing connections. The * marker is

there to show that the time is absolute, and this will be useful later. Equivalently, let $\vec{P}$ (resp. $\vec{T}/\vec{L}$) be the set of priorities (resp. periods/slot lengths) of the aforementioned connections. Periods are also often referred as intervals.

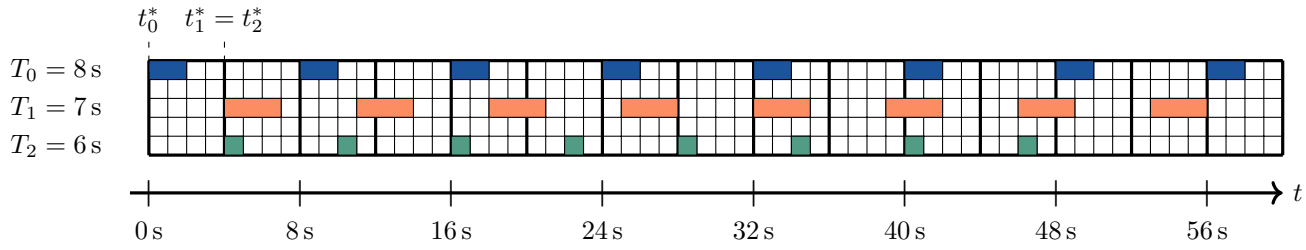


FIGURE 4.2: Example of 3 connection timings, with $P_0 = P_1 = P_2 = 1$.

A proposed cost function is presented at (4.1). This function is proportional to the number of collisions, $\#_{i,j}$, divided by the number of packets sent over a given connection, N. Since two connections do not always have the same number of packets in a given duration, the sum over the two connections is performed. The minimal duration that needs to be considered for counting the number of collisions and packets is the periodicity between the two connections. This duration can be computed as the Lowest Common Multiple (LCM) of T_i and T_j. As a reminder, the LCM and the Greatest Common Divisor (GCD) are linked by the following relationship: $\mathrm{LCM}_{i,j} \cdot \mathrm{GCD}_{i,j} = T_i \cdot T_j$.

$$C = \sum_{i,j>i} \#_{i,j} \left(\frac{P_i}{N_{i,j}} + \frac{P_j}{N_{j,i}} \right) \tag{4.1}$$

With that in mind, $N_{i,j}$ and $N_{j,i}$ can be expressed as a function of connections parameters:

$$N_{i,j} = \frac{\mathrm{LCM}_{i,j}}{T_i} = \frac{T_j}{\mathrm{GCD}_{i,j}} \tag{4.2}$$

It is essential to see that $N_{i,j}$ and $N_{j,i}$ are different as the order of the indices matters.
(4.1) now becomes:

$$C = \sum_{i,j>i} \#_{i,j} \left(\frac{T_i P_i + T_j P_j}{\mathrm{LCM}_{i,j}} \right) \tag{4.3}$$

From (4.3), it is clear the timing parameters $\vec{t}^*$ only influence, at most, the value $\#_{i,j}$. Given a new connection i, the role of the slot finder algorithm is to find the best t_i^* that minimises the partial cost C_i:

$$C_i = \sum_{j \neq i} \#_{i,j} \left(\frac{T_i P_i + T_j P_j}{\mathrm{LCM}_{i,j}} \right) \tag{4.4}$$

Example of cost computation

By taking the connections set depicted in FIGURE 4.2, one can compute the cost of such configuration:

$$C = \#_{0,1} \left(\frac{T_0 P_0 + T_1 P_1}{\mathrm{LCM}_{0,1}} \right) + \#_{0,2} \left(\frac{T_0 P_0 + T_2 P_2}{\mathrm{LCM}_{0,2}} \right) + \#_{1,2} \left(\frac{T_1 P_1 + T_2 P_2}{\mathrm{LCM}_{1,2}} \right) \tag{4.5}$$

$$= 4 \left(\frac{8 \cdot 1 + 7 \cdot 1}{56} \right) + 1 \left(\frac{8 \cdot 1 + 6 \cdot 1}{24} \right) + 3 \left(\frac{7 \cdot 1 + 6 \cdot 1}{42} \right) \tag{4.6}$$

$$= \frac{31}{12} \approx 2.6 \tag{4.7}$$

4.1.2 Efficiently computing the collisions

Until here, the challenge with $\#_{i,j}$ is that it does not currently have any analytic formula. If one wants to compute the number of collisions per period, one must count them one by one. For simple examples, such as with FIGURE 4.2, counting can be manually done, but this may become tedious as soon as N takes large values.

To simplify, a computer could go through each packet in connection j until period LCM is reached, and count each packet that is colliding with connection i:

$$\#_{i,j} = \sum_{n=0}^{N_{j,i}-1} \underbrace{\left(\overbrace{L_i}^{i \text{ ends}} > \overbrace{((t_j + nT_j) \bmod T_i)}^{j \text{ begins}} \right)}_{j \text{ begins before } i \text{ ends}} \vee \underbrace{\left(\overbrace{((t_j + nT_j) \bmod T_i)}^{j \text{ begins}} > T_i - L_j \right)}_{j \text{ ends after next } i \text{ begins}} \tag{4.8}$$

where $\vee$ is the logical OR.

In this, it is important to have $t_i \triangleq (t_i^* \bmod T_i) \leq t_j$. This is not a problem if $\vec{t^*}$ is kept sorted with respect to t_i, and if $t_j \triangleq \left(t_j^* - t_i\right) \bmod T_i$. One might ask: *why doing this?* This is done for multiple reasons: *(i)* it avoids checking if connection j starts before i, because it never happens in this case. *(ii)* This also reduces the number of branches that the program has to take: for more information on that, please refer to APPENDIX A.1. *(iii)* If $T_i \geq L_j$, a condition always met otherwise connection j would never have any time without collision, (4.8) only uses positive integers, making it very suitable with the fact that all the connections parameters can be represented as `unsigned int`. *(iv)* Lastly, having the timing anchors already sorted makes it very efficient to go through every connections without even checking which one starts first.

The bottleneck in (4.8) is the summation over $N_{j,i}$, a number that can be dramatically large. To further reduce the computation, one can observe the following property:

$$\begin{aligned}
(t_j + nT_j) \bmod T_i &= \Big((t_j \bmod T_i) + (nT_j \bmod T_i) \Big) \bmod T_i && (4.9)\\
&\quad \text{expressing } T_i \text{ and } T_j \text{ as functions of } \mathrm{GCD}_{i,j} \\
&= \Big((t_j \bmod T_i) + (nN_{j,i} \cdot \mathrm{GCD}_{i,j} \bmod N_{i,j} \cdot \mathrm{GCD}_{i,j}) \Big) \bmod T_i && (4.10)\\
&= \Big((t_j \bmod T_i) + m \cdot \mathrm{GCD}_{i,j} \Big) \bmod T_i && (4.11)\\
&\quad \text{with } m \in \{0, 1, ..., N_{i,j} - 1\} \\
&= (t_j \bmod T_i) \bmod \mathrm{GCD}_{i,j} + k \cdot \mathrm{GCD}_{i,j} && (4.12)\\
&\quad \text{with } k \in \{0, 1, ..., N_{i,j} - 1\} \\
&= \hat{t}_j + k \cdot \mathrm{GCD}_{i,j} && (4.13)
\end{aligned}$$

with $\hat{t}_j = (t_j \bmod T_i) \bmod \mathrm{GCD}_{i,j} = t_j \bmod \mathrm{GCD}_{i,j}$, see APPENDIX B.1.1 for proof of last identity, and each possible value of k is mapped exactly once over all the values of n. TABLE 4.1 shows an example of the ordering property of (4.13).

n	0	1	2	3
$(t_2 + nT_2) \bmod T_0$	4	2	0	6

k	0	1	2	3
$\hat{t}_2 + k \cdot \mathrm{GCD}_{0,2}$	0	2	4	6

TABLE 4.1: Unrolling all possible values for (4.9), with same parameters as connections 0 and 2 (FIGURE 4.2).

Finding the number of collisions reduces to finding the lower ($k_{\max}$) and upper ($k_{\min}$) collision bounds:

$$
\begin{aligned}
k_{\max} &= \underset{k}{\operatorname{argmax}} \left(L_i > \hat{t}_j + k \cdot \mathrm{GCD}_{i,j} \right) & k_{\min} &= \underset{k}{\operatorname{argmin}} \left(\hat{t}_j + k \cdot \mathrm{GCD}_{i,j} > T_i - L_j \right) \\
&= \underset{k}{\operatorname{argmax}} \left(\frac{L_i - \hat{t}_j}{\mathrm{GCD}_{i,j}} > k \right) & &= \underset{k}{\operatorname{argmin}} \left(k > \frac{T_i - L_j - \hat{t}_j}{\mathrm{GCD}_{i,j}} \right) \\
&= \left\lfloor \frac{L_i - \hat{t}_j}{\mathrm{GCD}_{i,j}} \right\rfloor_r - (r = 0) & &= \left\lfloor \frac{T_i - L_j - \hat{t}_j}{\mathrm{GCD}_{i,j}} \right\rfloor + 1
\end{aligned} \tag{4.14}
$$

with r the remainder of the current integer division. It is used to cut ties when the numerator is a multiple of the GCD: in those cases, k must be decreased because of the strict inequality sign.

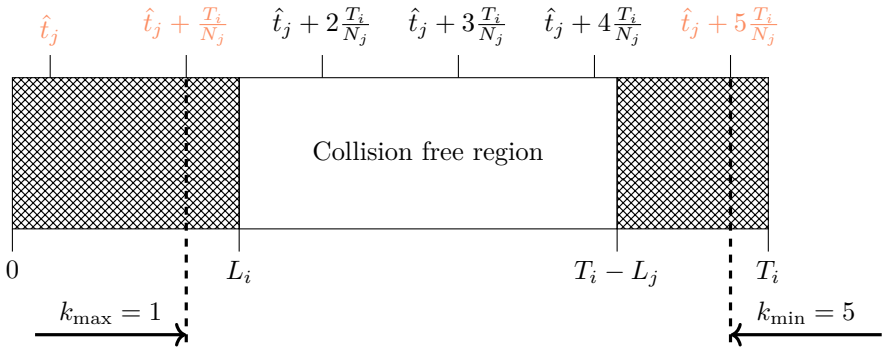


FIGURE 4.3: Illustration of collision bounds $k_{\max}$ and $k_{\min}$.

FIGURE 4.3 illustrates the fact that all timings can be brought back to a reference period T_i, from which one can directly obtain the number of collisions (in orange).

One problem with this method is when $L_i > T_i - L_j$: some collisions are counted twice since they appear in both collision regions. This is shown in FIGURE 4.4.

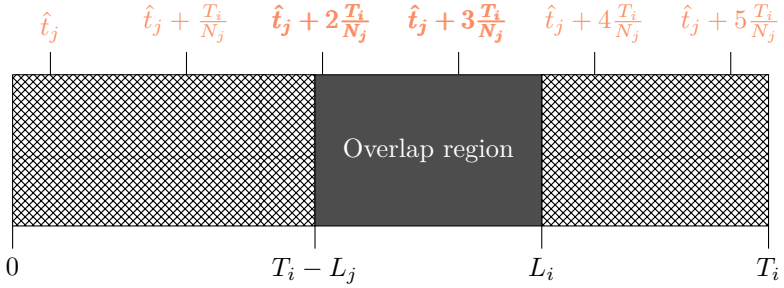


FIGURE 4.4: Illustration of collision bounds $k_{\max}$ and $k_{\min}$ where some collisions (in bold) are counted twice.

This eventual overlap is, however, not a real problem. The left and right operands in (4.8) can only both evaluate to 1 if $L_i + L_j < \min(T_i, T_j)$, which should never be the case as it would lead to a large number of collisions no matter what the slot finder chooses for the timing anchor.

Finally, the number of collisions can be expressed as:

$$
\#_{i,j} = (k_{\max} + 1)\,(k_{\max} > -1) + (N_{j,i} - k_{\min})\,(k_{\min} < N_{j,i}) \tag{4.15}
$$

This expression allows computing the number of collisions without requiring any loop. SECTION 5.4.1 conducts several tests to validate this metric.

C implementation

Listing 1 shows one implementation of the scoring function that is applied between two connections. After being summed over all pairs of connections, one will obtain the total score, C.

The code has been optimised to use as few divisions as possible, but using no division was impossible. Line 34 multiplies the overall score by $2^7 = 128$ to avoid loosing too much precision due to integer rounding. This product is almost equivalent to expressing the result in % ($\times 100$), but logical shifts are far more efficient than multiplications. This explains why a power of two was preferred.

```
1  typedef unsigned int uint;
2
3  uint score_two_connections(uint t1, uint interval1, uint duration1, uint prior1,
4                             uint t2, uint interval2, uint duration2, uint prior2) {
5      uint n_coll = 0;
6      uint gcd, kmin, kmax, rmax, n1, n2;
7
8      // GCD can be computed either with or without division operations
9      gcd = compute_gcd(interval1, interval2);
10     n1 = interval2 / gcd; // = lcm / interval1
11     n2 = interval1 / gcd; // = lcm / interval2
12
13     // Warning: t1 >= t2, otherwise it will not work
14     // Reminder: t1 := t1_star % interval1
15     t2 = (t2 - t1) % gcd;
16
17     // Here, some obvious optimisations are not done to preserve readability
18     // and any compiler should be able to optimise the follow code
19     if (duration1 + IFS > t2) {
20         kmax = (duration1 + IFS - t2) / gcd;
21         rmax = (duration1 + IFS - t2) % gcd;
22         n_coll += kmax + 1 - (rmax == 0);
23     }
24
25     if (interval1 > duration2 + t2 + IFS) {
26         kmin = (interval1 - duration2 - t2 - IFS) / gcd;
27         kmin += 1;
28         n_coll += (n2 - kmin)*(kmin < n2);
29     }
30     else {
31         n_coll += n2;
32     }
33
34     n_coll <<= 7; // This is done to avoid loosing precision due to rounding
35
36     return (n_coll*prior1)/n1 + (n_coll*prior2)/n2;
37 }
```

Listing 1: Implementation of the scoring function $C_{i,j}$.

4.1.3 The algorithm

Now that the metric of interest has been defined, an algorithm to find the best slot can be designed.

A straightforward – but inefficient – solution would be to skim through all possible parameters and find

the best ones. This procedure would always give the best solution but could also take a very long time. Since time is a critical constraint here, *i.e.*, the algorithm should run in less than 100 ms, one must take another approach.

As such, a novel algorithm has been developed with three goals in mind:

- **G1:** if it finds a good spot (*i.e.*, $\#_{i,j} = 0$), it takes it and stops searching;
- **G2:** it tries to minimise the time between connections;
- **G3:** it starts searching using, whenever it is possible, parameters from already existing connections.

FIGURE 4.5 presents how this method proceeds in an idealistic situation:

- **Step 1:** it adds connection 1 to the list;
- **Step 2:** it cannot use same period as connection 1, but connection 2 is added right next to connection 1 without inducing any future collision;
- **Step 3:** it can use the same period as connection 1, so it does, and connection 3 is added right next to connection 2 without inducing any future collision.

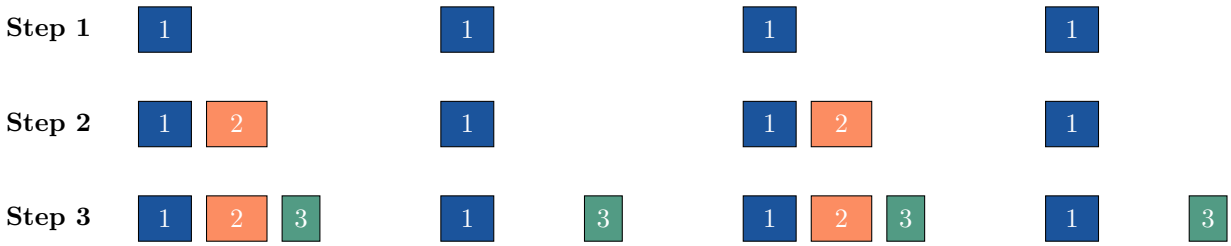


FIGURE 4.5: Slot finder ideal solution.

There are essentially two degrees of freedom that the algorithm can play with: the timing anchor and, in some cases, the period – also called the interval duration. While the timing anchor does not constrain its value, the interval can only be a multiple of 1.25 ms. For connection events, the value must also lie between 7.5 ms and 4 s. In practice, the slot finder will be asked to find an interval duration between minimal and maximal value, T_{min} and T_{max}. If one wants to impose the connection interval, then it can simply use $T_{\text{min}} = T_{\text{max}}$, see FIGURE 4.6.

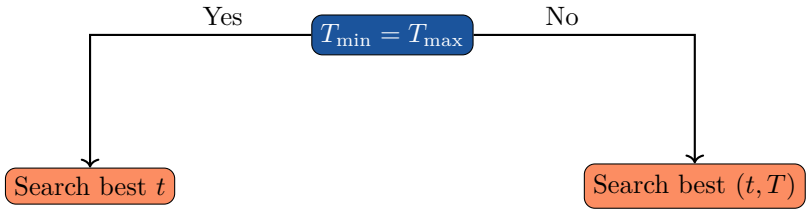


FIGURE 4.6: Slot finder algorithm: first decision.

The implementation of the left side of the decision tree is shown in LISTING 2. If the algorithm fails to find a good spot using all the parameters from previously added connections, it falls back to a random mode. This mode tries, for `MAX_TRIALS` number of times, randomly generated parameters that satisfy the constraints as mentioned earlier. Finally, the best scoring set of parameters is kept. Each parameter has its own Pseudo-Random Number Generator (PRNG), see LISTINGS 3 and 4. Those PRNGs are inspired by [27], and SECTION 5.4.3 describes their distribution properties.

```
if (range_interval == 0){ // Search best t
    interval = min_interval; // First, try to append connection at the end
    for (i = n_conn-1; i >= 0; i--) {
        t  = slots->t[i] + slots->duration[i] + IFS; // End of i-th conn.
        t %= interval;

        score = eval_insert_conn(slots, t, interval, duration, prior);
        if (score == 0) { // Update best score and break
            /* ... */
        }
        else if (score < best_score) { // Update best score
            /* ... */
        }
    } // Then, if did no find : random search for MAX_TRIALS
    while ((n_trials < MAX_TRIALS) && (best_score > 0)) {
        t = xorshift_anchor(&rng_state, interval);

        score = eval_insert_conn(slots, t, interval, duration, prior);
        if (score == 0) { // Update best score and break
            /* ... */
        }
        else if (score < best_score) { // Update best score
            /* ... */
        }
        n_trials++;
    }
} // Else, search best (t, T) ...
```

Listing 2: Slot finder algorithm: searching for best t part.

```
/*
 * XORSHIFT
 * used to generate random interval values (in us) from 0 to max_value
 * returned value is a multiple of INTERVAL_STEP
 */
uint xorshift_interval(state *rng_state, uint max_value){
    uint x = rng_state->x;
    x ^= x << 7;
    x ^= x >> 9;
    x ^= x << 8;
    rng_state->x = x;

    // First the PRN is MOD-ed, then rounded to nearest multiple of step.
    x %= max_value;
    x += INTERVAL_STEP - 1;
    x -= x % INTERVAL_STEP;
    if (x > max_value) return 0;
    return x;
}
```

Listing 3: Pseudo-random interval generator.

```c
/*
 * XORSHIFT
 * used to generate random timing anchor values (in us) from 0 to max_value
 */
uint xorshift_anchor(state *rng_state, uint max_value){
    uint x = rng_state->x;
    x ^= x << 7;
    x ^= x >> 9;
    x ^= x << 8;
    rng_state->x = x;

    return x % max_value;
}
```

LISTING 4: Pseudo-random anchor timing generator.

Even though those PRNGs can be very efficiently implemented, relying on randomness does not certify that it will find the best solution possible. Instead of trying random interval durations, it may be wiser to try periods that are multiples (or divisors) of previously used values. Nonetheless, adding such heuristics to the current implementation will increase the time complexity.

A proposed solution can mitigate this problem by pre-processing interval bound values (see FIGURE 4.7).

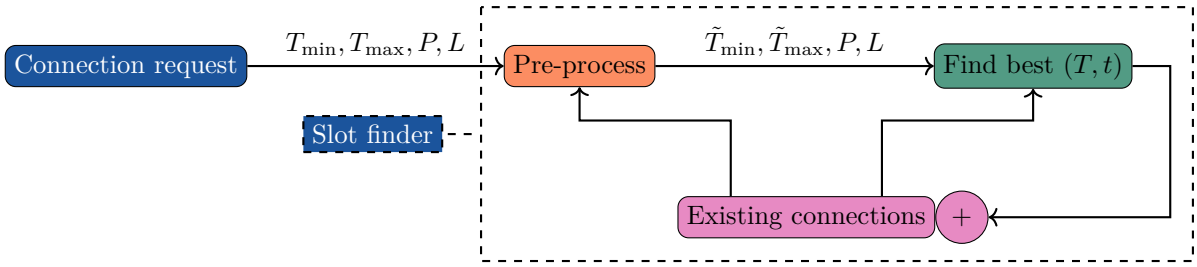


FIGURE 4.7: Slot finder algorithm: pre-processing information.

The role of the pre-processing node is to adapt the input parameters, $T_{\min}$ and $T_{\max}$, to increase the chances of finding a good slot. In the case of the present algorithm, this is done by updating minimal and maximal interval values to, when possible, match a multiple of the other interval values. An ideal case is when the pre-processing finds an interval value, $\tilde{T}$ that is a multiple (or divisor) of every other interval value already used and that falls in the range $[T_{\min}, T_{\max}]$. In this case, it will change the parameters to $\tilde{T}_{\min} = \tilde{T}_{\max} = \tilde{T}$. Nonetheless, determining common multiples is an expensive task as it requires many division and multiplication operations. The possible benefits obtained by the pre-processing part highly depend on all connections parameters and is very hard to quantify.

Therefore, SECTION 5.4.4 provides benchmarks of the algorithm without any pre-processing. The goal is to analyse both best and worst-case scenarios to observe the algorithm's limits. As time optimisation is vital, simulations show that the algorithm always runs under 100 ms in the worst case possible, at the lowest clock frequency.

4.2 Filtering data

In many digital applications that include some data acquisition through sensors, it is convenient to filter the input data (see Figure 4.8).

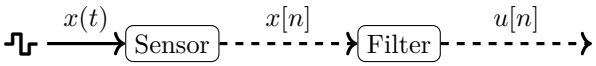


Figure 4.8: External and continuous data $x(t)$ is first converted into a digital and discrete signal, then filtered out.

The role of this filter is often to damp high frequency noise effects, but it can also be manifold. Here, the filter should act as a memory and keep previous information – like a capacitor – during a certain time. To this end, a Infinite Impulse Response (IIR) filter is used:

$$u[n+1] = \left(1 - \frac{1}{2^k}\right) u[n] + x[n] \tag{4.16}$$

The latter is trivial to implement in `C` code and presents another advantage: it is very efficient in integer arithmetic as it only requires bit-shifts and additions (Listing 5). Actually, the inner part of the function can execute in 3 clock cycles, which takes 187.5 ns at 16 MHz (minimal clock speed). The parameter k will be referred to as the smoothing parameter since changing its value will modify the damping effect of the filter.

```c
int filter(int u, int x, int k) {
    return (u + x) - (u << k);
} // Also works with unsigned integers as u > (u << k)
```

Listing 5: `C` implementation of filter (4.16).

Using a recurrence relation for $u[n]$ does not highlight all the filter's properties. Therefore, solving for $u[n]$ gives:

$$u[n] = \left(1 - 2^{-k}\right)^{n-1} \left(\sum_{i=0}^{n-1} \left(1 - 2^{-k}\right)^{-i} x[i] \right) + u[0] \left(1 - 2^{-k}\right)^{n} \tag{4.17}$$

where $u[0]$ is known. See Appendix B.2.1 for solution details.

4.2.1 Constant input signal

In the case where $x[n]$ is constant, *i.e.*, $x[n] = x$, the solution simplifies:

$$u[n] = \left(1 - 2^{-k}\right)^{n} \left(u[0] - 2^k x\right) + 2^k x \tag{4.18}$$

One interesting property of this filter is that, regardless of the initial condition $u[0]$, the output value will tend toward $2^k x$:

$$\lim_{n\to\infty} u[n] = \underbrace{\left(1 - 2^{-k}\right)}_{0<\cdot<1}^{\infty} \left(u[0] - 2^k x\right) + 2^k x \tag{4.19}$$

$$= 0 \cdot \left(u[0] - 2^k x\right) + 2^k x \tag{4.20}$$

$$= 2^k x \tag{4.21}$$

As such, one must keep in mind the magnitude of the output signal $u[n]$ will grow with 2^k. Therefore, it may be interesting to normalise the output signal to later compensate for this factor.

4.2.2 Other properties

When analysing the properties of a given filter, two significant tests are often conducted: the impulse response and the frequency response.

For the filter impulse response, *i.e.*, when input signal is $x[n] = \delta[n-1]$, one can obtain it using the constant form of the filter (4.18) with initial condition $u[0] = 1$ and constant input signal $x = 0$:

$$u[n] = \left(1 - 2^{-k}\right)^{n-1} \tag{4.22}$$

for $n > 0$.

FIGURE 4.9 shows the impulse response for various smoothing parameters k. Here, the effect of the smoothing parameter is clear: the larger k, the longer the signal will be memorised. Increasing k can also have the drawback of increasing the response time of one system. Finding the best k is, therefore, a matter of trade-offs. As a reminder, a value $k = 0$ is equivalent to not filtering the input signal at all.

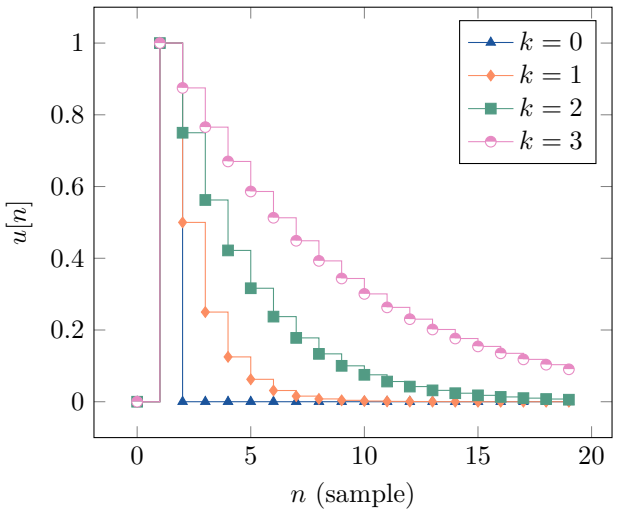


FIGURE 4.9: Filter impulse response for various smoothing parameters k.

A simple way to determine the transfer function, $H(z)$, is to use the properties of the Z-transform:

$$u[n+1] = \left(1 - \frac{1}{2^k}\right) u[n] + x[n] \tag{4.23}$$

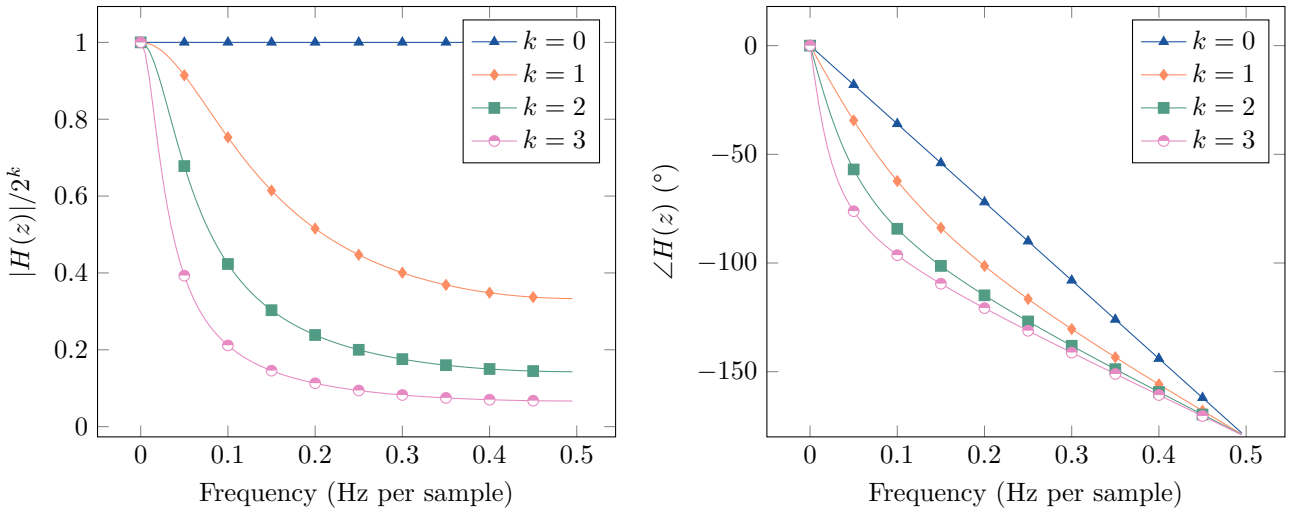
$$\Downarrow \mathcal{Z}\{f[n]\} \tag{4.24}$$

$$zU(z) = \left(1 - \frac{1}{2^k}\right) U(z) + X(z) \tag{4.25}$$

$$U(z)\left(z - \left(1 - 2^{-k}\right)\right) = X(z) \tag{4.26}$$

Which finally gives:

$$H(z) = \frac{U(z)}{X(z)} = \frac{1}{z - (1 - 2^{-k})} \tag{4.27}$$



(A) Magnitude

(B) Phase

FIGURE 4.10: Filter frequency response for various smoothing parameters k.

FIGURE 4.10 shows the filter frequency response for various smoothing parameters k. As shown, the damping effect grows exponentially with k, which means that the range of values for k is minimal before the signal becomes completely filtered out.

4.3 Channel Assessment

The role of the channel assessment task has already been presented in previous Sections. This Section will first skim through an example of algorithm implementation from NXP. Some details are being intentionally not covered here for non-disclosure agreement reasons. Then, an upgrade to the method is proposed to further enhance interoperability with Wi-Fi.

The main idea behind their algorithm is to give each channel a quality estimator based on the number of times a packet has failed to be received, divided by the total amount of packets sent over some duration. This indicator gives an estimate of the PER for each channel. Then, this estimate goes through some filter.

Each channel is initially given a perfect score, and, using the filtered PER, it will be increased or decreased depending on if the PER falls above or below a certain threshold. If the score of a channel has reached a too low value, it is removed from the channel map. Channels that are not in the channel map are not used in the AFH sequence and no longer considered by this algorithm.

Lastly, to keep a minimal number of active channels, the device can promote some removed channels in a trial mode. This trial mode will add the channels to the channel map but with a lower score than they were given at the start. Channels in probation mode are regularly chosen from the set of unused channels, but their RSSI level can prevent promoting channels used by other devices.

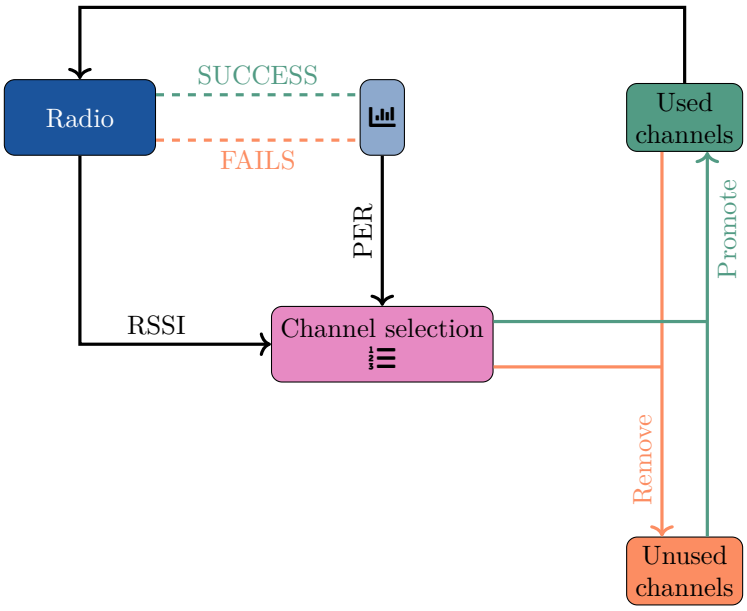


Figure 4.11: Simplified diagram of the channel assessment algorithm presented in Section 4.3.

4.3.1 Problem definition

One downside of the approach mentioned above is that it only allows the PER to remove channels from the map, assuming this is sufficient for many BLE applications. While this is mainly fine for BLE on its own, this can cause problems when the BLE device becomes the dominant emitter on a channel and can interfere with other technologies using the same frequencies. In this case, the current algorithm will only discover that other protocols use the same frequencies if it causes its PER to increase, but otherwise not.

This unidirectional methodology means that BLE can become the source of interference and deteriorate the communications of other protocols. Wi-Fi, a prevalent technology, is one of the protocols that can suffer from this side-effect. The goal is to enhance the current algorithm to detect Wi-Fi signals, disregarding their power. This detection will no longer only use an error-based approach but also based on the RSSI level.

4.3.2 Objectives

On top of what is already done, the enhanced algorithm should detect the strongest Wi-Fi signal and remove the BLE channels that are overlapping with its frequency bands. Only the strongest Wi-Fi signal is kept as, in real cases scenarios, there can be multiple Wi-Fis in the neighbouring area (FIGURE 4.12), and removing all used frequency bands would not leave much room for BLE. Therefore, the goal is to avoid interference with the primary Wi-Fi signal, which is assumed to be the one in the same flat/room/house as the BLE device.

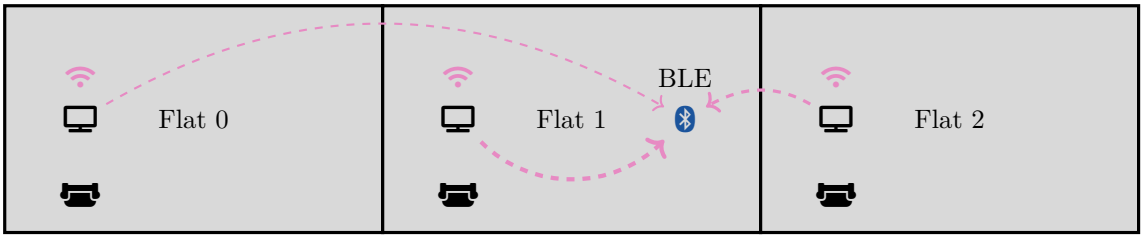


FIGURE 4.12: BLE device detecting surrounding Wi-Fi signals, where line thickness represents sensed power.

In the example of FIGURE 4.12, the BLE device scans multiple signals (FIGURE 4.13), but it should detect that Wi-Fi channel 6 is the dominant one. The algorithm should also identify Wi-Fi channels that are overlapping between each other.

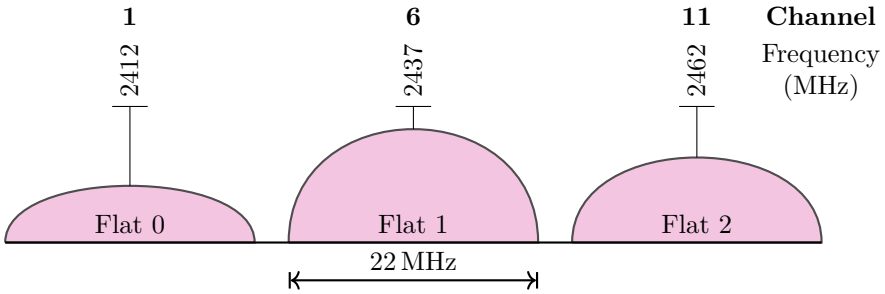


FIGURE 4.13: Wi-Fi spectrum sensed by BLE device in FIGURE 4.12.

FIGURE 4.14 highlights the modifications in the algorithm required to account for Wi-Fi channel detection. The content of the Wi-Fi detection block is further detailed below, but it is important to mention that it should only return one channel index. This index should indicate the closest BLE channel from the centre frequency of the Wi-Fi. If no Wi-Fi is detected, this value can either be -1 or any other invalid channel index.

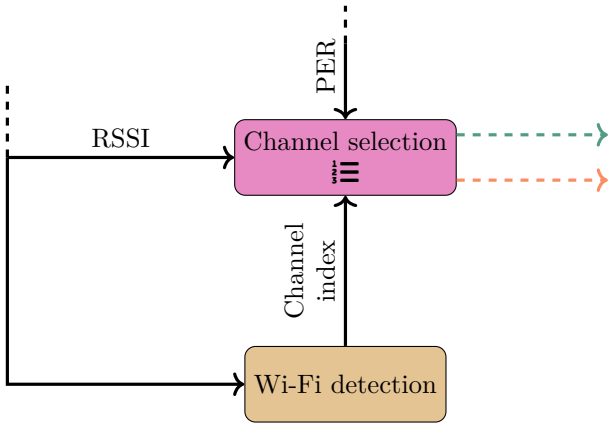


FIGURE 4.14: Enhancement of channel assessment algorithm.

4.3.3 Centre frequency detection

To better identify the Wi-Fi channels, the proposed method tries to determine the centre frequency of each channel. Once this frequency is found, the affected BLE channels can be determined using a simple mapping between BLE and Wi-Fi channels. For example, a peak at 2437 MHz corresponds to Wi-Fi channel 6, which overlaps with BLE channels 11 to 19 if the Wi-Fi channels are 22 MHz wide. If one desires to find the dominant Wi-Fi signal, one can assume that it will be located where the most significant power is. The vector $\vec{\text{RSSI}}$ contains the RSSI level of each BLE channel. For every scan performed, the RSSI status is updated by taking the maximal value sensed. Taking the maximum value is important here to detect the Wi-Fi as quickly as possible. Nonetheless, this method is sensitive to other signals (not Wi-Fi) that could have a higher power but narrower frequency bands, such as ZigBee. To better capture the Wi-Fi signal, and not others, a convolution over frequencies is performed (4.28). Convoluting has two significant advantages: *(i)* it better captures centres via symmetric weights, and *(ii)* it gives more importance to broader frequency bands, eliminating narrower signals such as ZigBee's. The result, $\vec{p}$, is somewhat related to the power spectrum of the Wi-Fi channels.

$$\vec{p} = \vec{\text{RSSI}} * \vec{w} \tag{4.28}$$

with $\vec{w} = \begin{bmatrix} 1 & 2 & 4 & 2 & 1 \end{bmatrix}$.

The choice of $\vec{w}$ was motivated by multiple reasons. *(i)* Using symmetrical weights that are powers of two allows for a very efficient implementation (Listing 6). The latter runs in exactly 921 clock cycles, which takes 57.5625 µs at 16 MHz. This duration is comparable to the overhead time taken for radio acquisition (100 µs to 200 µs). On average, the method takes 25 clock cycles (1.5625 µs) per channel. *(ii)* The weights are bigger in the centre to better capture the centre of the frequency bands. Lastly, *(iii)* the symmetry in the weights is a consequence of the channels' spectrum being symmetric around its central frequency.

```c
void convolve(int RSSI[37], int p[37]) {
    for (int i=2; i < 35; i++)
        p[i] =  (RSSI[i-2] + RSSI[i+2])
             + ((RSSI[i-1] + RSSI[i+1]) << 1)
             + ( RSSI[i  ]              << 2);
    // Boundaries
    p[ 0] =  (      0  + RSSI[ 2])
          + ((      0  + RSSI[ 1]) << 1)
          + ( RSSI[ 0]             << 2);
    p[ 1] =  (      0  + RSSI[ 3])
          + ((RSSI[ 0] + RSSI[ 2]) << 1)
          + ( RSSI[ 1]             << 2);
    p[35] =  (RSSI[33] +       0 )
          + ((RSSI[34] + RSSI[35]) << 1)
          + ( RSSI[35]             << 2);
    p[36] =  (RSSI[34] +       0 )
          + ((RSSI[35] +       0 ) << 1)
          + ( RSSI[36]             << 2);
}
```

Listing 6: C implementation of the convolution (4.28), equivalent to `numpy.convolve(a, v, mode="same")`.

Since values in $\vec{\text{RSSI}}$ are only updated once at a time, it is not necessary to recompute the complete convolution, but only part of it. This optimisation further reduces the amount of computation needed. The partial convolution (Listing 7) executes in 90 to 120 clock cycles (5.5625 µs to 7.5 µs), roughly the same number of cycles per channel as previous implementation. However, in addition to computing the convolution, this method also determines the location of the new peak value. If the peak frequency stays the same, an invalid index (-1) is returned.

```c
/*
 * RSSI: old RSSI measurements
 * p, max_p: old convolution results and maximum value
 * new_RSSI, index: new RSSI measurement at channel [index]
 */
int convolve_partial(int RSSI[37], int p[37], int max_p, int new_RSSI, int index) {
    int diff = new_RSSI - RSSI[index];
    int index_max_p = -1; // -1 means index_max_p did not change from previous
    RSSI[index] = new_RSSI;

    index += 2; // index + 2
    if (index < 36) {
        p[index] += diff;
        if (p[index] > max_p) {
            max_p = p[index];
            index_max_p = index;
        }
    }
    index -= 4; // index - 2
    if (index > 0) {
        p[index] += diff;
        if (p[index] > max_p) {
            max_p = p[index];
            index_max_p = index;
        }
    }
    index += 3; // index + 1
    diff <<= 1; // diff << 1
    if (index < 36) {
        p[index] += diff;
        if (p[index] > max_p) {
            max_p = p[index];
            index_max_p = index;
        }
    }
    index -= 2; // index - 1
    if (index > 0) {
        p[index] += diff;
        if (p[index] > max_p) {
            max_p = p[index];
            index_max_p = index;
        }
    }
    index += 1; // index
    diff <<= 1; // diff << 2
    p[index] += diff;
    if (p[index] > max_p) {
        max_p = p[index];
        index_max_p = index;
    }
    return index_max_p; // Index of (possibly) new maximum p
}
```

LISTING 7: C implementation of the partial convolution (4.28), where only one input value changes at each computation.

4.3.4 Scanning strategy

As Section 3.2.5 mentions, it is preferable to only scan one BLE channel per scan interval. If the scan interval is 10 ms long, a common value, this means that there is a 370 ms gap between two measurements of the same channel. However, neighbouring channels should experience similar behaviour as one Wi-Fi channel spans over multiple consecutive BLE channels. This assumption is one of the reasons that lead to the use of Frequency Hopping for BLE communications. As such, FH is also used here.

Assuming that the ISM spectrum can fit three non-overlapping Wi-Fi channels (see Section 2.2), a *hop value* of 13 was chosen so that no two consecutive scans are testing the same potential Wi-Fi channel.

This approach may not be robust against highly varying Wi-Fi channels but, in the case of this thesis, it is assumed that the Wi-Fi channels are relatively constant over time, at least over a few hours. The latter is a safe assumption as Wi-Fi protocol does not implement FH. The variation of the strongest Wi-Fi signal is mainly linked to multiple Wi-Fi routers or turning on and off the access point. In real applications, such events do not happen frequently enough to make the algorithm unusable.

Last but not least, the RSSI is itself noisy: it contains the background noise, but it is also highly dependent on the moment the device measures the signal. Therefore, to reduce the impact of high-frequency variations, the RSSI measurements are filtered with the same filter as in Section 4.2. Later in the document, Section 5.5.2 discusses with details the benefits and drawbacks of using a filter, in different scenarios.

4.3.5 Threshold-based detection

The presence of background noise imposes the use of the threshold value, `threshold`, to limit false positive detection. The use of a filter and the convolution by non-normalised weights increase the practical value of this threshold. If the Wi-Fi signal is known to always be above N, then a Wi-Fi channel is considered to be active if $p >$ `threshold`:

$$\texttt{threshold} = \underbrace{2^k}_{\text{Filter}^1} \cdot \underbrace{10}_{\text{Convolution}^2} \cdot N \tag{4.29}$$

Otherwise, if the maximal value, $p_{\max}$, falls below this threshold, an invalid channel index is returned to indicate that no Wi-Fi signal is present.

4.3.6 Wi-Fi channel detection

Figure 4.15 summarises the required operations to perform Wi-Fi channel detection. As a whole, the implementation of this enhancement should not add more than 10 µs to 20 µs at 16 MHz to the current algorithm execution. After filter, RSSI values are marked with a *tilde*. Section 5.5 provides multiple tests to assess the robustness of present enhancement and shows encouraging results that could lead to implementation in state-of-the-art products.

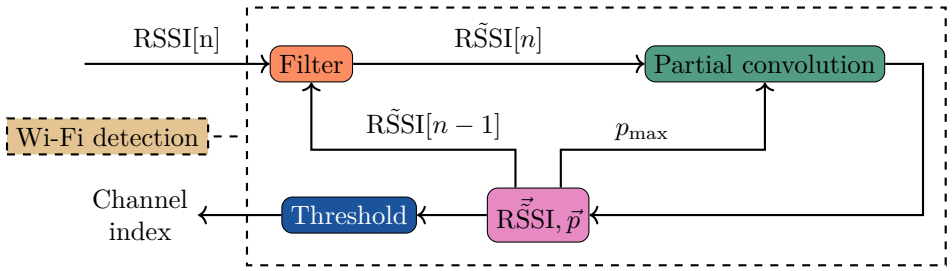


Figure 4.15: Wi-Fi detection block.

[1] The gain of the filter transfer function for constant input signal.
[2] $10 = 1 + 2 + 4 + 2 + 1$, *i.e.*, the sum of the window coefficients.

5 Measurements

The previous Chapters introduced the key components on which the BLE technologies are built and the challenges that modern applications leverage. A significant part when designing solutions to problems consists of simulations and measurements. The present Chapter will therefore describe the various simulations and tests conducted for this thesis. Due to limited access to physical facilities, simulations were the principal source for measurements. However, few measurements on real hardware were taken from NXP's employees' previous tests.

5.1 Clock drift and collisions

Measuring the drift between can become quite tedious as the drift, in itself, is not something absolute. Indeed, the drift quantity depends on the reference frame that is chosen. Using a third device to measure the drift between two other devices will give a value that depends on the third device's clock. This dependence is why knowing the exact absolute drift of one clock may not be especially interesting. Instead, it is easier to use the relative drift, ρ_r, between two clocks:

$$\rho_r = \rho_1 - \rho_2 \tag{5.1}$$

As it will be seen below, only the relative drift is required when analysing collisions between two connections.

The present scenario, inspired by the measurements taken, focuses on the analysis of two ACL connections. As earlier described, ACL connections only send data once in a while, *i.e.*, in opposition to a continuous stream, but often last for hours or days. One basic example would be a 3-devices setup: a phone, a tablet and a TV. The TV is the peripheral of two BLE connections, one for each other device. Devices are, respectively, called D_{P}, D_{T} and D_{TV}. In between two messages, devices are running on a 32 kHz sleep clock to save power. In often cases, the precision of this clock is much worse than the active clock used during packet exchange and is therefore considered the main source of frequency drift. As D_{TV} must correct any frequency drift it has with the two other clocks, the problem will be when D_{P} drifts relatively to D_{T}. Their ideal clock period is assumed to be T_{N} but their clocks are drifting independently by ΔT_{P} and ΔT_{T}. Both connections can have different periodicity $T_{\mathrm{ACL,P}}$ and $T_{\mathrm{ACL,T}}$, different data packets lengths L_{P} and L_{T}, and are initially separated with a delay d. This is presented in Figure 5.1.

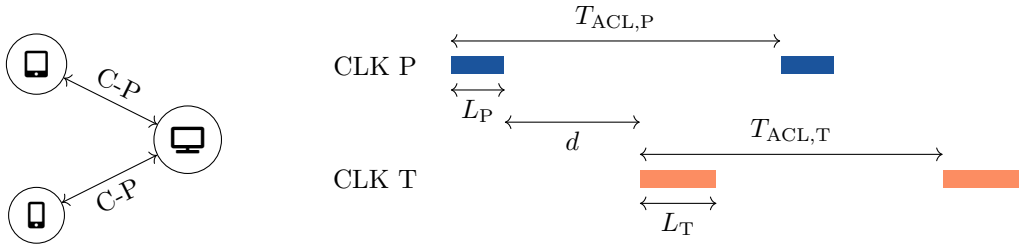


FIGURE 5.1: Two ACL connections scenario.

Here, T_{ACL} must not be mistaken with T. T is the sleep clock period, fixed after manufacturing the device, while T_{ACL} is the period of the ACL connection, which often ranges from 50 ms to 250 ms.

5.1.1 A first simple case

First, this scenario will be further simplified to equal ACL periods and equal packet lengths. In other words, $T_{\text{ACL,N,P}} = T_{\text{ACL,N,T}} = T_{\text{ACL,N}}$ and $L_{\text{P}} = L_{\text{T}} = L$. Using what has been done in SECTIONS 3.2.1 and 3.2.2, the difference in "speed" can be written as:

$$\begin{aligned}
|V_{\text{P}} - V_{\text{T}}| &= |(T_{\text{ACL,N,P}} + \Delta T_{\text{ACL,P}}) - (T_{\text{ACL,N,T}} + \Delta T_{\text{ACL,T}})| \\
&= |n\,(T_{\text{N}} + \Delta T_{\text{P}}) - n\,(T_{\text{N}} + \Delta T_{\text{T}})| \\
&= n\,|\Delta T_{\text{P}} - \Delta T_{\text{T}}| \\
&= n\left|\frac{\rho_{\text{T}}}{f_N\,(1+\rho_{\text{T}})} - \frac{\rho_{\text{P}}}{f_N\,(1+\rho_{\text{P}})}\right| \qquad (5.2) \\
&\approx n\left|\frac{\rho_{\text{T}}}{f_N} - \frac{\rho_{\text{P}}}{f_N}\right| \\
&\approx n\frac{|\rho_{\text{T}} - \rho_{\text{P}}|}{f_N} = T_{\text{ACL,N}}\,|\rho_{\text{T}} - \rho_{\text{P}}| \qquad (5.3)
\end{aligned}$$

Therefore, the time until the first collision is:

$$t_c^1 \approx \frac{d}{T_{\text{ACL,N}}\,|\rho_{\text{T}} - \rho_{\text{P}}|} T_{\text{ACL,N}} = \frac{d}{|\rho_{\text{T}} - \rho_{\text{P}}|} \qquad (5.4)$$

And one can apply very similar reasoning to find the collision duration in seconds:

$$t_{\Delta c} \approx \frac{2L}{|\rho_{\text{T}} - \rho_{\text{P}}|} \qquad (5.5)$$

5.1.2 Different packet lengths

Taking constant packet lengths in ACL connections is not very realistic since the packet length changes depending on the information to be sent. Most of the time, packets will be empty, and, only once in a while, a non-empty packet will be sent containing status information or else.

5.1.3 Different period lengths

Having an equal period duration for all connections can be a good solution if it is possible to fit all of them in a timing window while maximising the distance between each link. When there are too many connections, this may become a caveat: if periods are almost the same, then the relative drift between connections will be slight. Therefore, a collision might last very long. A solution to this problem is to take period durations that have the largest LCM possible.

5.1.4 About PER

As described earlier, the packets are subject to errors and can be lost. In most cases, this means that if a packet is lost, then it is going to be retransmitted during the next packet event. If the first packet of communication is lost, then the peripheral will not send any acknowledgement. Nonetheless, these side effects are not taken into account here.

5.1.5 Validation with measurements

The test setup, described above, contains two ACL connections with a period of 10 ms. The reference connection (see FIGURE 5.2) sends two messages during one interval, while the second connection only sends one. The packets filled with colour are sent by the central, and the peripheral device sends the empty ones. Those packets are essentially acknowledgement packets.

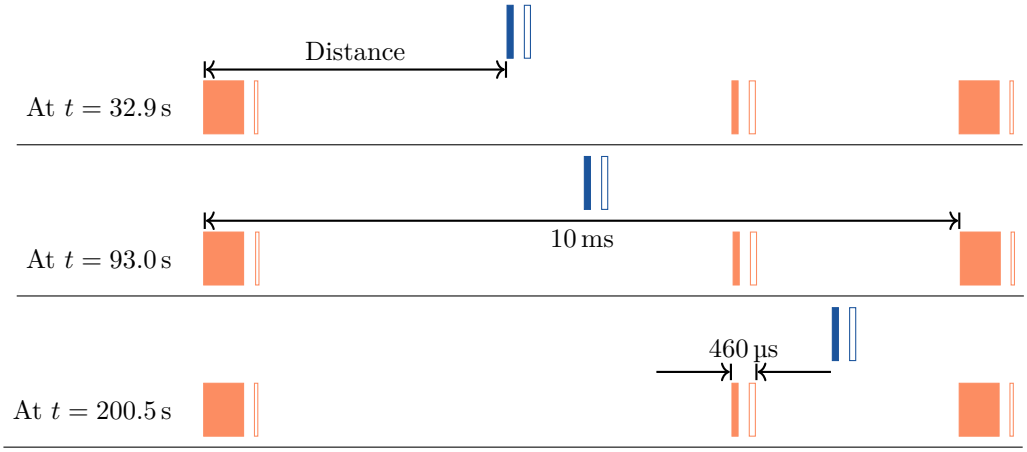


FIGURE 5.2: Evolution of the relative timing anchors, with orange connection as reference.

Even though the packets do not all have the same length, the first collision between the two connections will happen on packets that have the same length and same period. It is then possible to use (5.4) and (5.5) to estimate the collision timings. The only parameter that is to be determined is the relative clock drift.

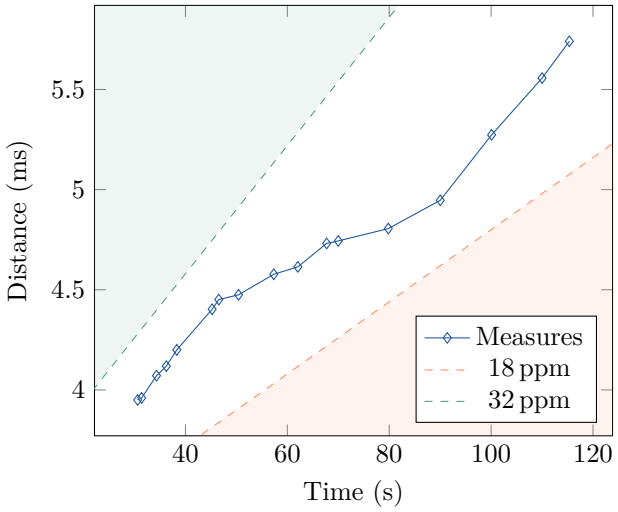


FIGURE 5.3: Relative clock drift between two ACL connections over time.

Figure 5.3 plots the evolution of the distance between the two packets over time. By computing the slope between distance and time, one can estimate the relative drift to be in the range 18 ppm to 32 ppm. From that, it is possible to estimate t_c^1 and $t_{\Delta c}$ and compare them to actual values (see Table 5.1).

Quantity	Estimate	Measure
t_c^1	50 s to 90 s	88.00 s
$t_{\Delta c}$	65 s to 115 s	82.66 s

Table 5.1: Comparison of collision estimates and measures.

The results show that the estimates provide a good idea of the order of magnitude of collisions timing. Determining the clock drift with precision can be challenging, and one can use a range of possible values instead.

Finally, the time spent in a collision is larger than the interval by orders of magnitude. If nothing is done, that is more than 8000 packets that will be lost on each side. The effect of clock drift is therefore not negligible. The quality of communication relies on an excellent mechanism to alternate between the different connections. Such a tool will diminish the time without collision from 8000 packets to a few.

5.2 Channel Assessment

As discussed in previous Sections, the role of the CA algorithm is to evaluate the 37 BLE data channels. After evaluation, it should only keep the good channels to provide low PER communications.

A common way to observe the performance of one consolidation algorithm is to run tests with and without the algorithm. These tests help to understand the way the channels are selected and how quickly the algorithm can adapt. Of course, the results mainly depend on the simulation scenario. As a way to provide robust analysis, numerous simulations are run under multiple different cases.

As observing all available measurements could take a lot of time and space, this report will only analyse two different scenarios and their results. Section 5.2.1 compares the impact of using CA and not using CA at all (NOCA). Then, Section 5.2.2 analyses what happens when multiple Wi-Fi channels are sequentially activated.

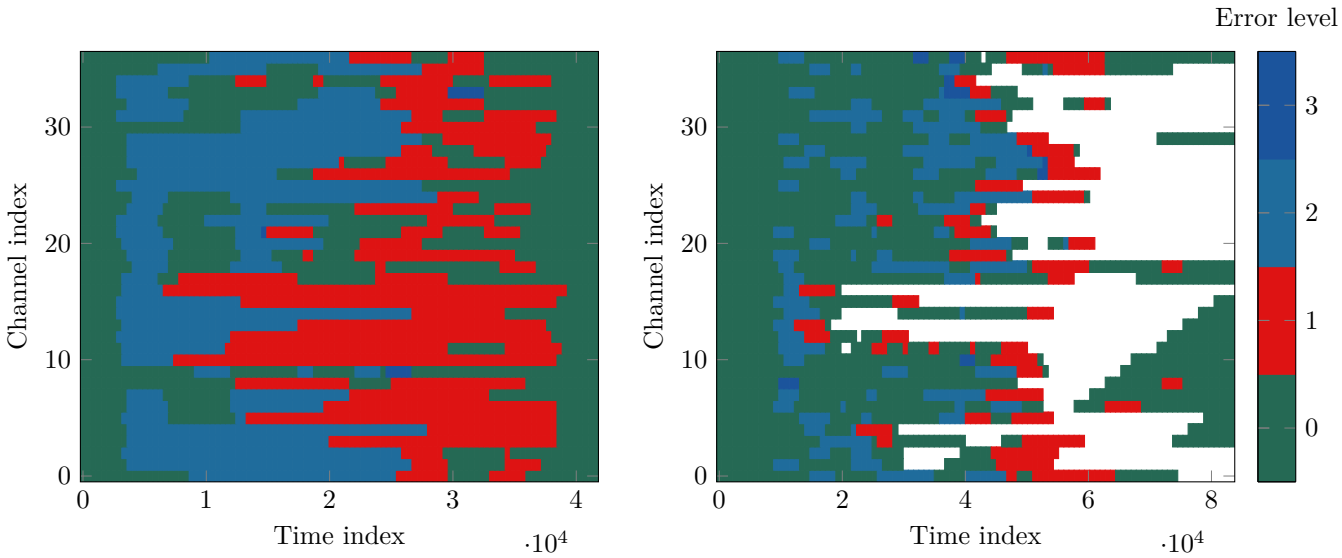
The measurements were taken by NXP using an earlier version of the CA algorithm. Regarding some non-disclosure agreements, the information has been anonymised, and details on the different error statuses have been obfuscated. As the measurements contain billions of data points, most of the data displayed below has been post-processed. It reduced the said information to a minimum. It is therefore essential to keep in mind that there is much more information in reality.

Hence, this short analysis aims to give a glimpse of the possible benefits that CA can bring.

5.2.1 CA versus NOCA

This part contains tests, first without CA (left), then with CA (right), both with similar conditions. Those tests consisted of moving the BLE emitter further away from the receiver and returning to the initial position. At the same time, some devices used Wi-Fi signals to transmit on the same frequencies as various BLE channels. The duration and the sampling frequency of both tests are not mainly the same, but it is not relevant here.

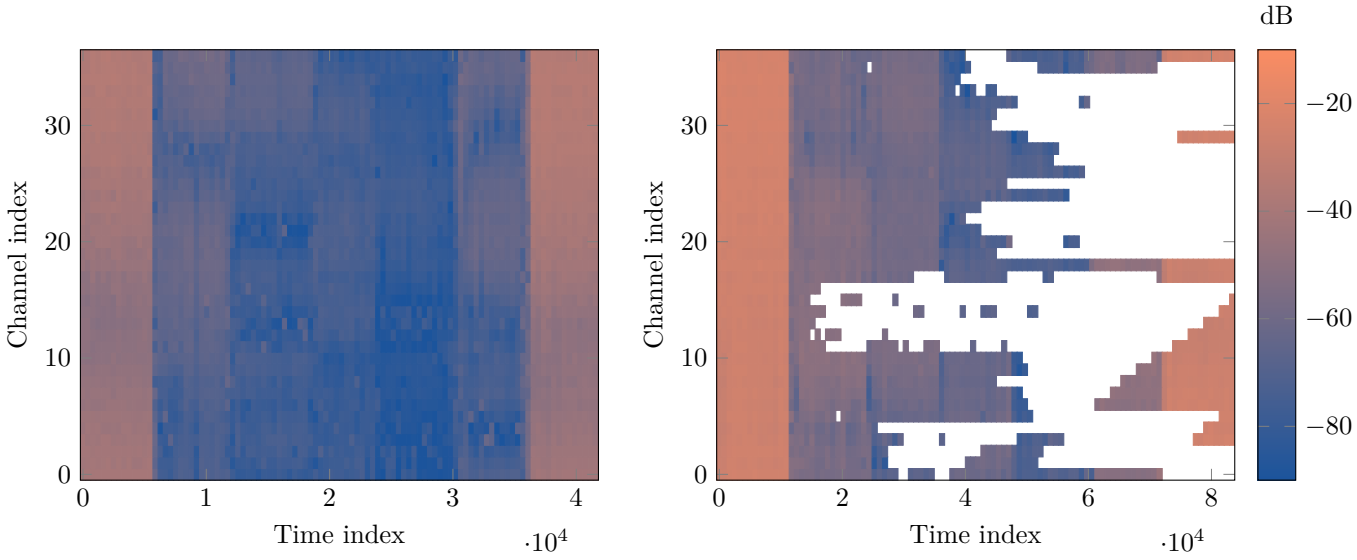
Figure 5.4 describes the different packet statuses that are received over time. Any level other than zero indicates an error. Again, this analysis will not cover what the error types are, as it is not relevant. In Figure 5.4a, one can see that there are many errors and that nothing is done to avoid using the bad channels. It is an extreme example where most BLE channels are affected by errors. In opposition to NOCA, Figure 5.4b shows how the CA algorithm tries to remove the poor BLE channels. When a channel is removed, it is represented by blank spaces. One can see that not every error causes a channel to be removed but rather the accumulation of them over time.



(A) NOCA (B) With CA

FIGURE 5.4: Time evolution of packet statuses over the different BLE channels.

FIGURE 5.5 shows the different RSSI levels that are sensed over time. FIGURE 5.5a highlights well the fading caused by the displacement of the emitter. One can also observe that all the BLE channels do not have the same RSSI level at a given time instant. This difference can be, for example, the consequence of reflection and diffraction, physical phenomenons that depend on the frequency. FIGURE 5.5b, in adequation with 5.4b, shows that BLE channels that were removed did not have their RSSI level measured. Nonetheless, BLE devices can still measure the RSSI level of black-listed channels. As discussed in SECTION 4.3, the CA algorithm can measure the RSSI level of unused channels to choose some to promote. Here, only the used channels are shown.

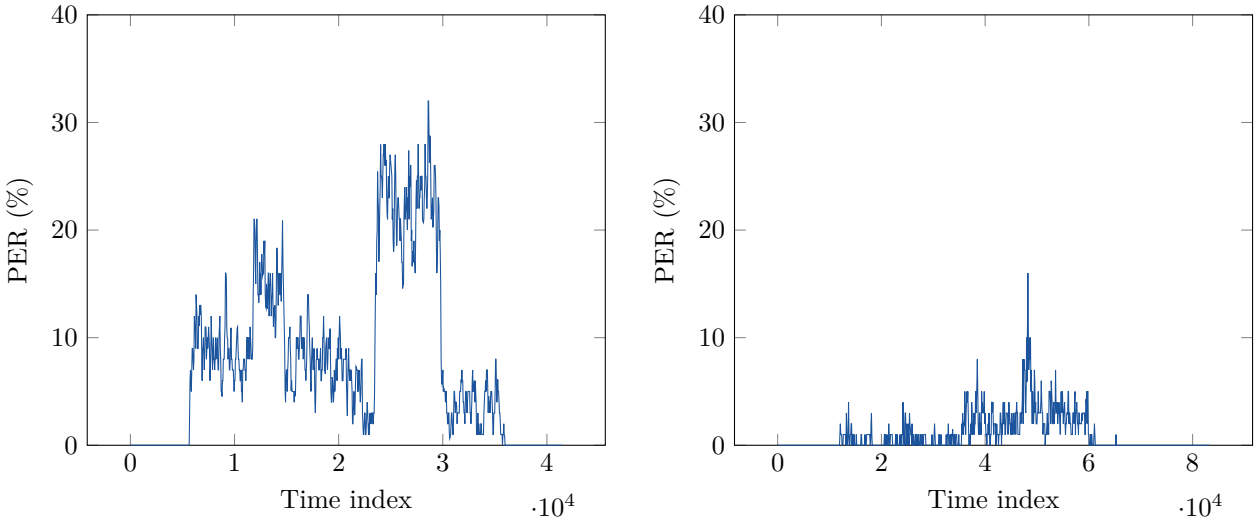


(A) NOCA (B) With CA

FIGURE 5.5: Time evolution of BLE channels RSSI levels.

In FIGURE 5.6, one can observe the evolution of the average PER over time. This average is computed over all the used BLE channels. The various sudden PER increases are aligned with the peaks in packet errors in

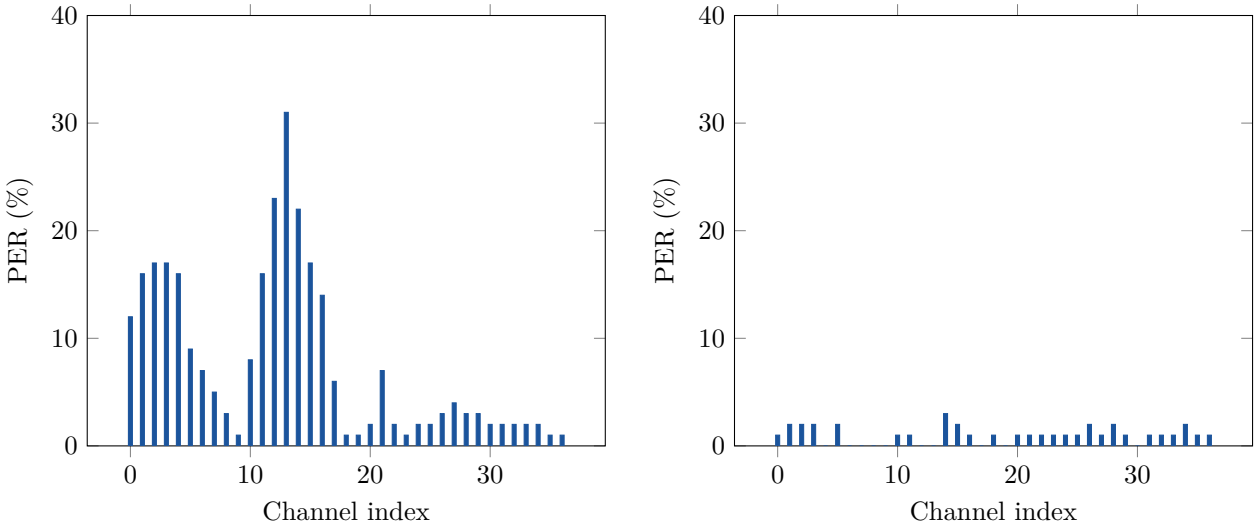
Figure 5.4. Between two peaks, Figure 5.6a shows a more-or-less constant PER. The absence of CA leads to a PER that mostly lies in the range 10 % to 25 %. When the goal is to keep a PER below 10 %, NOCA is not sufficient. On the other hand, Figure 5.6b emphasises the benefits of using CA as the PER drops below 5 % during most of the test duration.



(a) NOCA (b) With CA

Figure 5.6: Time evolution of average PER.

Finally, Figure 5.7 helps to understand which BLE channels were the most impacted by errors. The resulting PER is, therefore, a time average. Here, the benefits of using CA are also clear when comparing Figures 5.7a and 5.7b.



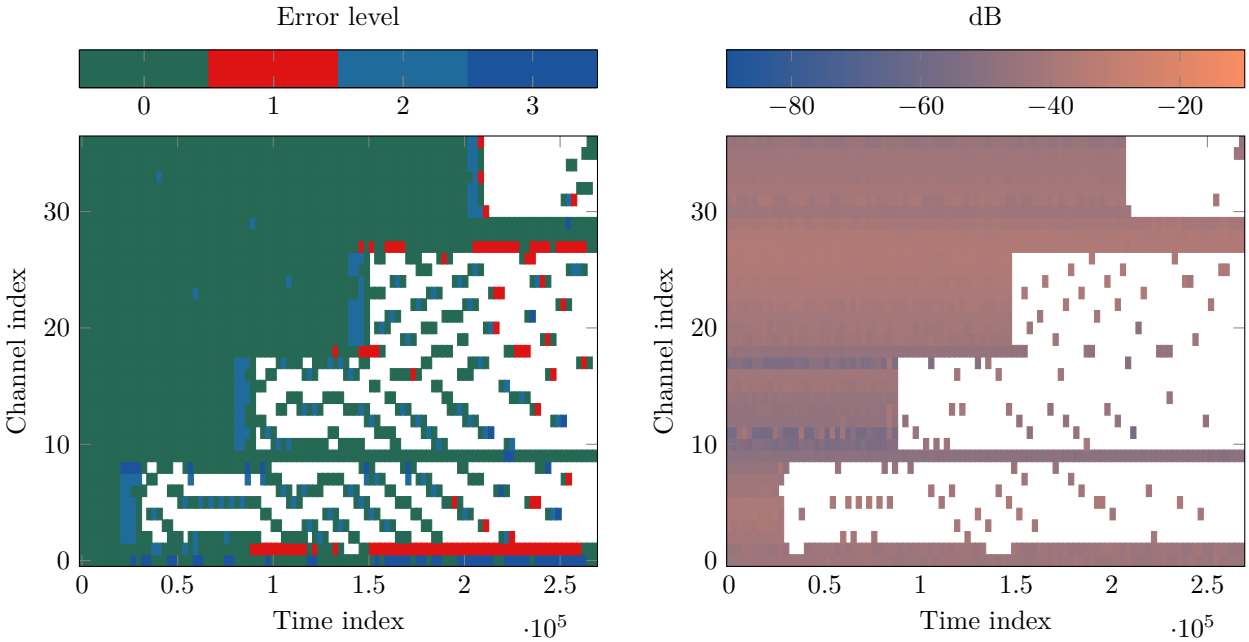
(a) NOCA (b) With CA

Figure 5.7: Average PER for each BLE channel.

5.2.2 Impact of multiple Wi-Fi channels

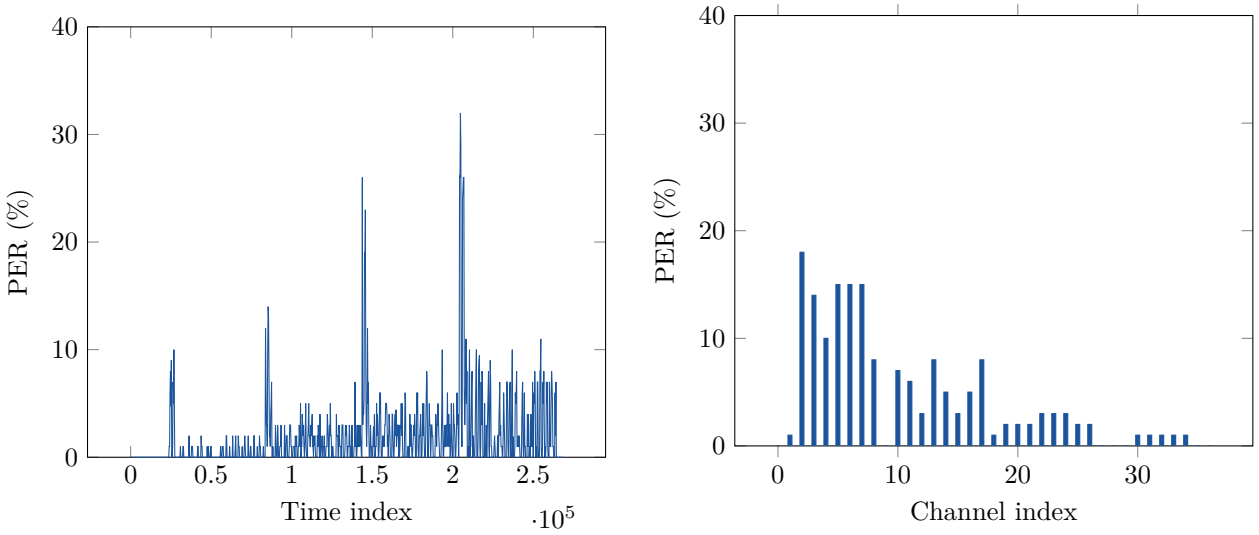
Another relevant test to conduct is to observe what happens when multiple Wi-Fi channels active simultaneously. The more Wi-Fi channels are activated, the fewer BLE channels will be evaluated as usable. Nonetheless,

as previously discussed, it is not possible to remove all channels. A minimal number of active channels has to be kept. This minimum is defined by the device designer and will impact the CA performance. If this minimum is less than the number of good channels, then the CA algorithm will keep some bad channels active. Keeping lousy channels active can increase the PER. Here, it is assumed that all BLE channels could have been removed when necessary.



(A) Status (B) Heatmap

FIGURE 5.8: Time evolution of packet statuses and RSSI levels.



(A) PER over time (B) PER per channel

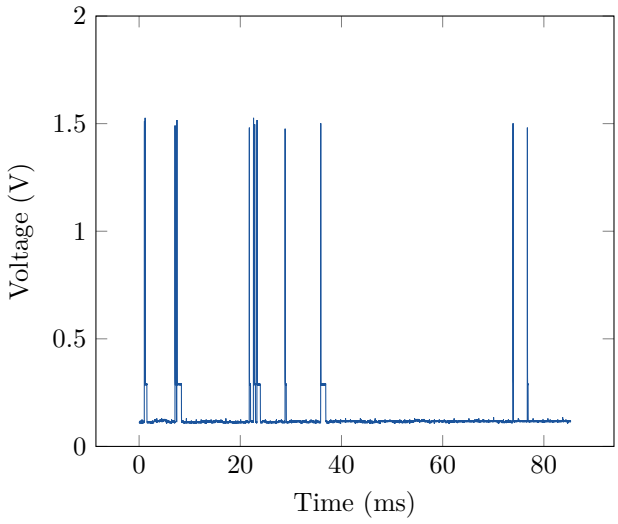
FIGURE 5.9: Average PER over time and per BLE channel.

In FIGURE 5.8, one can clearly distinguish the different timings at which the Wi-Fi channels are activated. FIGURE 5.9 shows the average PER as function time and BLE channel. As expected, the sudden activations

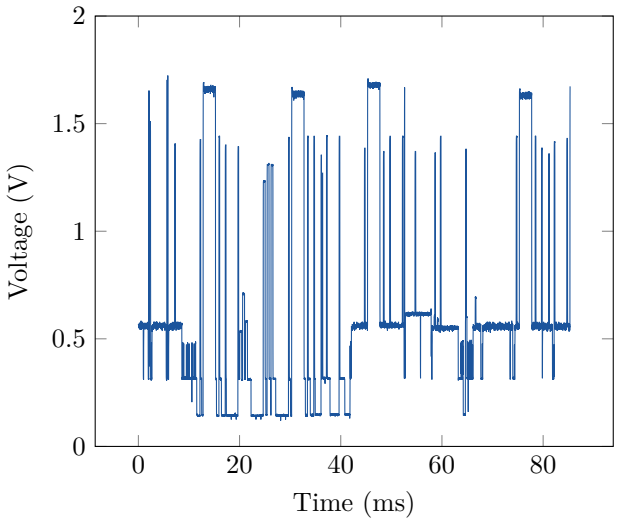
of Wi-Fi channels cause the PER to increase (see FIGURE 5.9a) rapidly. However, the CA algorithm performs well, and the PER is quickly lowered. As a result, the average PER stays below 10 %.

5.3 Wi-Fi activity

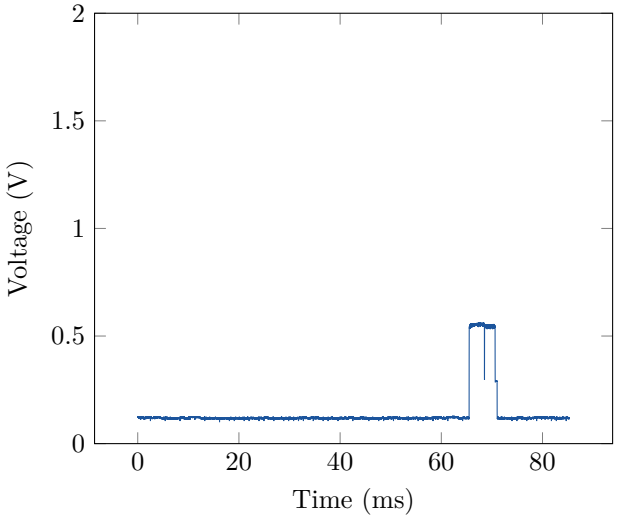
As introduced in SECTION 3.2.5, detecting Wi-Fi activity highly depends on when the scanning window occurs and on the Wi-Fi activity itself. It is easy to understand that the more devices use a given Wi-Fi, the easier it is to detect its signal. Multiple measurements have been conducted to quantify the average activity of a domestic Wi-Fi connection. Those were made in different rooms of the same house: *(i)* the basement, *(ii)* the office, and *(iii)* the garage. During the three tests, the Wi-Fi was used to perform an Internet speed test with a laptop, and the signal was recorded at 3.125 MHz with the iPerf tool. The environment was also crowded with other devices using Wi-Fi.



(A) Basement



(B) Office

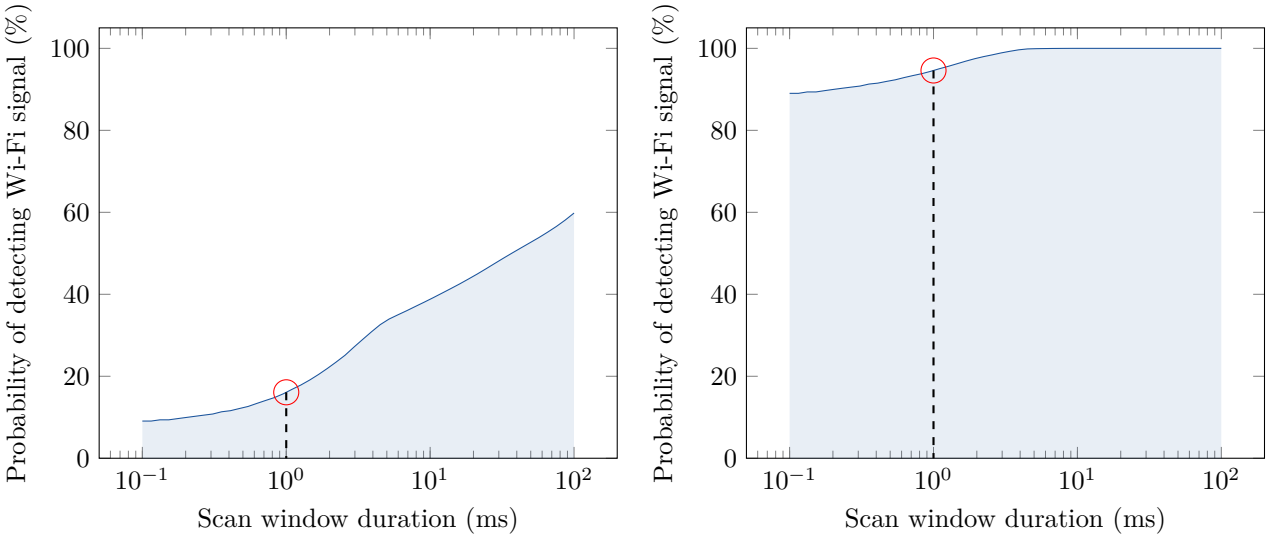


(C) Garage

FIGURE 5.10: Wi-Fi activity in different rooms in a home environment.

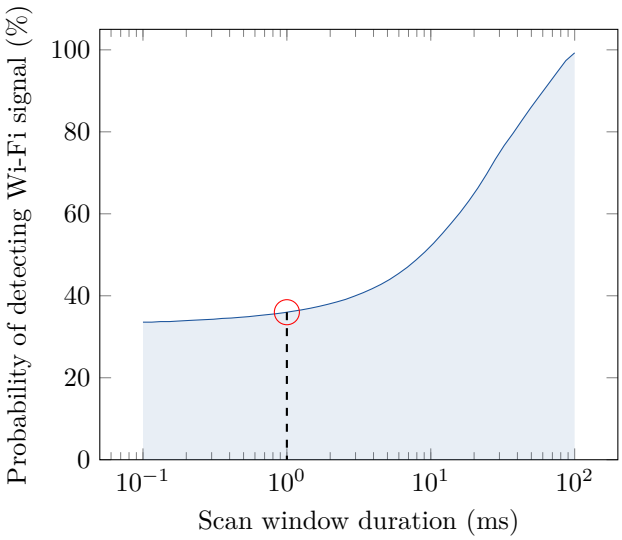
FIGURE 5.10 describes the activity in the different rooms with the RSSI level of the Wi-Fi signals measured in Volt. In those measurements, the noise level is around 0.1 V and the range of values for Wi-Fi spans from 0.2 V to 1.7 V. The different levels are mainly due to different devices with varying emitting power capabilities. FIGURE 5.10b depicts the high activity from surrounding devices. As one could expect, the garage has the lowest activity since this room is the most isolated one. Here, all the signals are emitted by the same Wi-Fi channel. However, it is not necessarily always true, and the opposite results in a lower Signal to Noise Ratio (SNR). In other words, identifying Wi-Fi channels in a crowded spectrum can become troublesome.

By first converting the analog signal (FIGURE 5.10) into digital, then convoluting it with different size scan windows, one can obtain FIGURE 5.11. The latter illustrates, for every room, the probability of detecting a Wi-Fi signal in function of the time that is spent scanning. Again, the office, *i.e.*, the room with the highest activity, has the biggest probability of detecting Wi-Fi signals for a given time window. Nonetheless, the two other rooms do not offer such high probabilities. One can see that, for a 1 ms long scan, the detection probability ranges from 15 % to 95 %. As the typical scan duration is around 800 µs, the simulations in SECTION 5.5 will cover probabilities from 10 % to 100 %.



(A) Basement

(B) Office



(C) Garage

FIGURE 5.11: Probability of detecting Wi-Fi activity in different rooms in a home environment.

5.4 Slot finder benchmarks

The slot finder implementation relies upon multiple building blocks, and it essential to validate their correctness. One must also recall that correctness is not the only goal here: the method must execute in less than 100 ms.

The following Sections discuss those subjects in details by applying the concepts to practical examples.

5.4.1 Testing and validating the number of collisions

To test and validate the expression proposed in (4.15), a graphical interface, see Figure 5.12, has been developed to manually check if that the predicted number of collisions matches the actual one. Then, a series of assertion-based tests has been written to verify the correctness further.

The blue bars represent the reference connection, *i.e.*, the one which starts first. On the other hand, the orange bars represent the second connection that will be added. The idea of this graphical interface is to highlight the impact of the different parameters the slot finder can play with. In addition to those parameters, *i.e.*, the interval duration T and the timing anchor t, one can also modify the slot length, L.

The number of collisions, displayed on the top right corner, is the number of times the blue and orange bars are overlapping, in between the two red vertical lines. This way, it is very easy to verify the correctness of the number of collisions formula.

A last interesting behaviour to observe is the impact of the parameters' values on the LCM, and therefore on $N_{0,1}$ and $N_{1,0}$, the numbers in front of T_0 and T_1. For example, a small variation in the interval duration due to the interval PRNG can potentially lead to a very large LCM value.

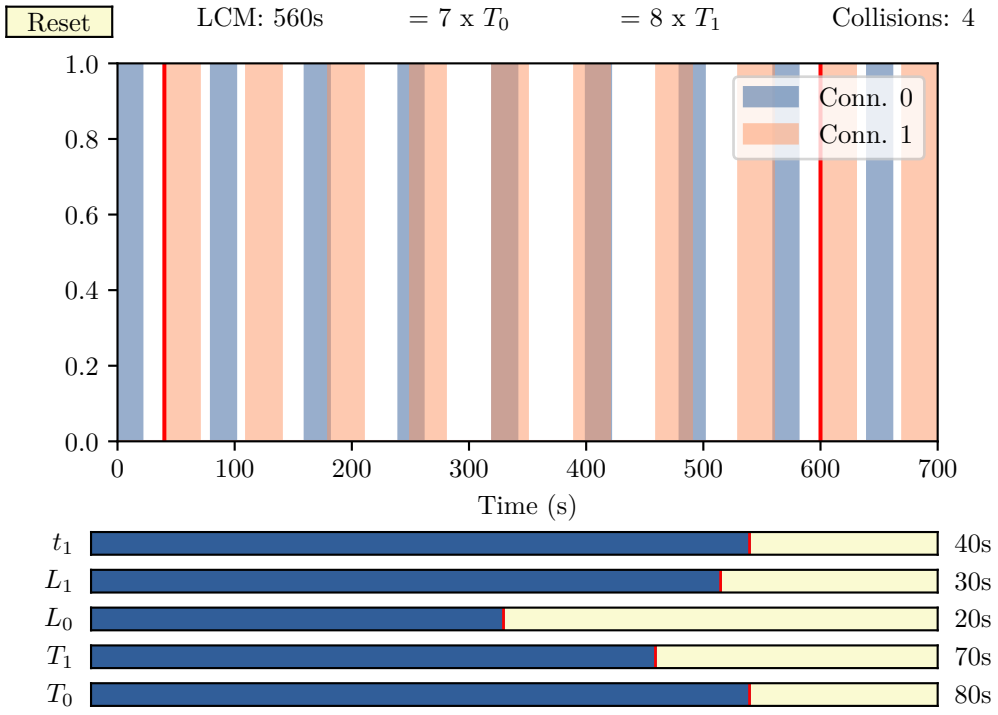


Figure 5.12: Interface developed for number of collisions validation. The two red lines define the LCM interval between the two connections. See Appendix A.2 for the code that was used to create this interface.

5.4.2 Example of collision bounds

The current Section features details on one of the essential concepts behind the slot finder: the collision bounds. The idea is to map the timing of all events into a reference frame, using nothing more than the modulo operand. As detailed in Section 4.1.2, this mapping allows to express all the events (packets) in an ordered sequence. This order is no more related to the time at which those packets are sent but rather to their relative position to the reference connection. In two of the three cases, connection 0 starts first, explaining why the reference duration is equal to $T_0 = 8\,\text{s}$ (Figure 5.13).

Figure 5.14 details how the collision bounds work when applied to values from Figure 5.13. For every pair of connections, the one which starts the first[1] is defined as the reference connection. Each tick on the top horizontal axis represents the start of one packet of the second connection. The left crosshatched zone indicates the first packet from the reference connection, and the right zone's width is the same as the packet width from the second connection.

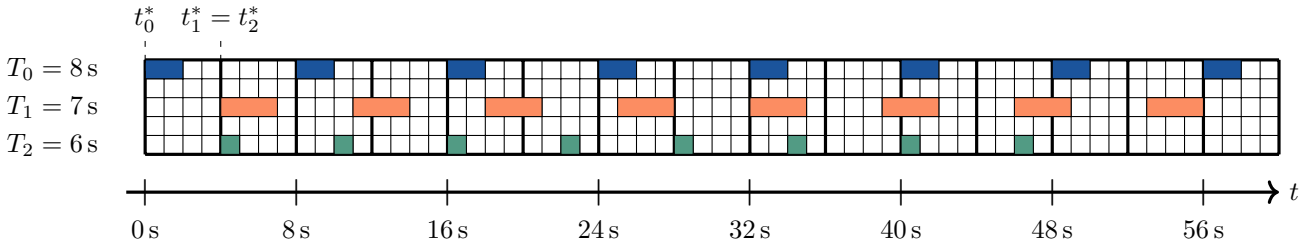


Figure 5.13: Example of 3 connection timings, with $P_0 = P_1 = P_2 = 1$ (bis).

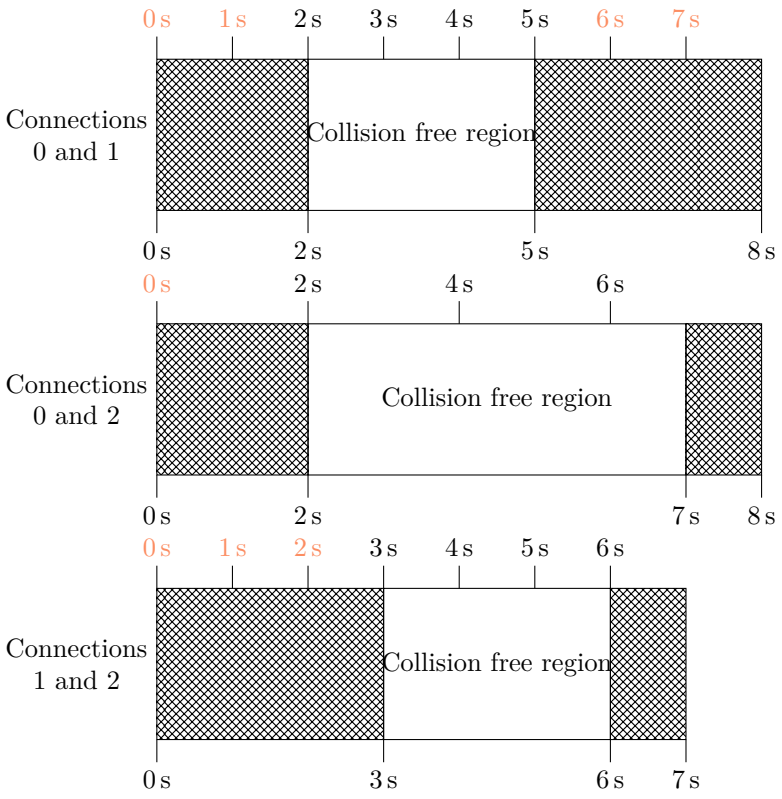
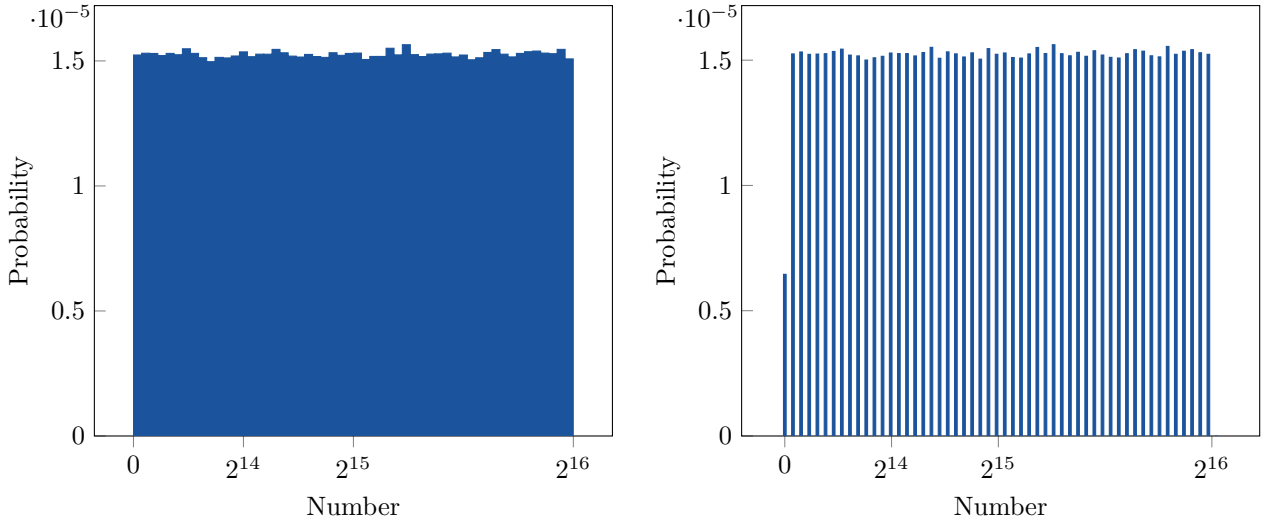


Figure 5.14: Example of collision bounds, based on connections set from Figure 5.13.

[1] As a reminder, $t_i \triangleq \left(t_i^* \bmod T_i\right)$ is used to determine the timing anchor of a connection, not the absolute timing t_i^*. Refer to Section 4.1 for more details.

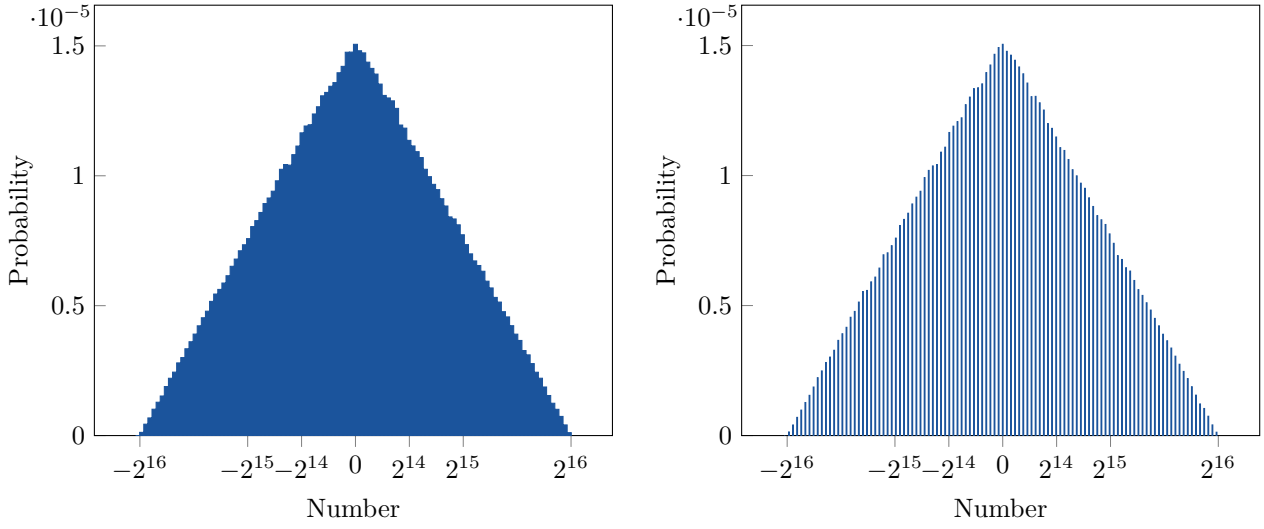
5.4.3 PRNG distributions

In the context of this algorithm, the two PRNGs have two objectives: *(i)* generate uniformly distributed numbers, and *(ii)* with the shortest execution time possible. From the timing point of view, xorshifts PRNGs are very efficient as they only use logical operators. Here, the modified versions use modulo operations which increases the number of cycles to generate one random number, but it cannot be avoided in this case. Nonetheless, the next Section discusses the timing impact of generating random numbers. For the uniform distribution part, Figures 5.15 and 5.16 describe, respectively, the distribution of the numbers generated and of the difference between consecutive numbers. For the right part, the space between bins highlights the fact that the interval PRNG can only generate multiples of 1.25 ms. The latter creates a near-uniform distribution: rounding towards the nearest multiple causes the value 0 to appear less frequently than others (Figure 5.15b).



(a) Anchor timing (b) Interval

Figure 5.15: Distribution of the pseudo-random value generated using the xorshift algorithm (16 first bits).



(a) Anchor timing (b) Interval

Figure 5.16: Distribution of the difference between two consecutive pseudo-random values generated using the xorshift algorithm (16 first bits).

5.4.4 Slot finder benchmark

In an effort to evaluate the performances of the slot finder algorithm in terms of speed of execution, FIGURE 5.17 details the execution of multiple simulations. Those simulations range from the best to the worst case possible. A perfect situation occurs when the algorithm finds a solution that scores 0 – the lowest score possible – from the first trial. On the other hand, the worse case will try every possible set of parameters it is allowed to test without ever finding ones that lead to a perfect score. Both extremes are essential to estimate the minimal and maximal bounds of the algorithm. The lower bound has a $\mathcal{O}(n)$ complexity and the upper bound follows a $\mathcal{O}(n^3)$ curve, see FIGURE 5.18 to better observe them.

Between those limits, the algorithm's time complexity varies depending on the degrees of freedom as well as on the number of random trials allowed. The different test results shown in FIGURES 5.17 and 5.18 are encoded using a C-D-R abbreviation for conciseness. TABLE 5.2 describes how to decode the different test parameters.

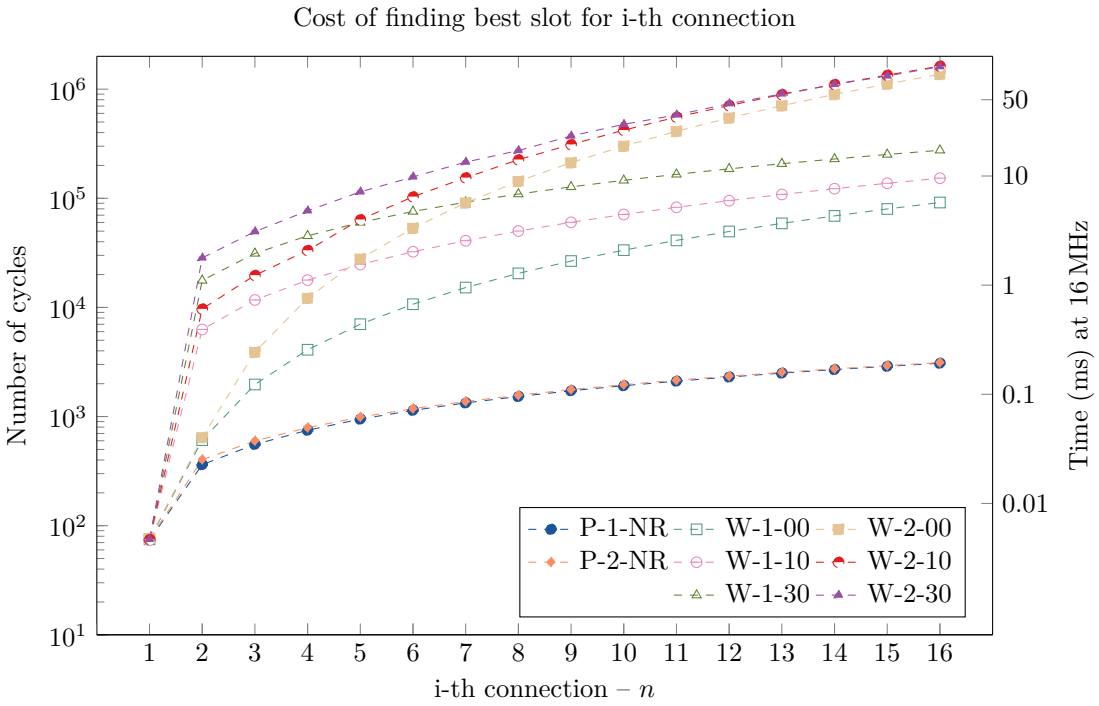


FIGURE 5.17: Benchmark of the slot finder algorithm in various extreme cases in $x - \log y$ scale.

Letter	Description	Value	
C	Case	Perfect (P)	directly finds perfect score
		Worst (W)	never finds perfect score
D	Degrees of freedom	1	$T_{\min} = T_{\max}$, searches best t
		2	$T_{\min} \neq T_{\max}$, searches best (t, T)
R	Max. # of random trials	00 – 10 – 30 – Non Relevant (NR)	

TABLE 5.2: C-D-R Decoding table for the tests used in FIGURES 5.17 and 5.18.

The take-away message from those charts is that, in the worst-case scenario, the execution time stays below 100 ms. However, applications with more than 5 or 6 connections are rare and, taking this into account, the algorithm will most likely always execute under 10 ms.

One can see that the cost of additional random trials becomes negligible as the number of connections

increases. In particular, the principal difference in time of execution comes from the increase from one to two degrees of freedom. Therefore, the importance of the pre-processing step is vital to reduce the complexity by already determining good candidates for the interval. If the pre-processing can find a good interval value – such that no search on that parameter is required after – the algorithm will then execute under 1 ms for eight connections or less.

In practice, these assumptions are not so far from reality. The number of connections in many applications is often limited to the few devices one person owns: a smartphone, a speaker, a tablet, some earbuds and a computer. This number is, of course, not a hard limit but, a personal wireless network, the subject of this thesis, rarely contain more than 5 or 6 connections. Also, many BLE devices engage multiple connections that are similar. The variety of interval values that are chosen is therefore limited, and finding a good interval candidate can become straightforward.

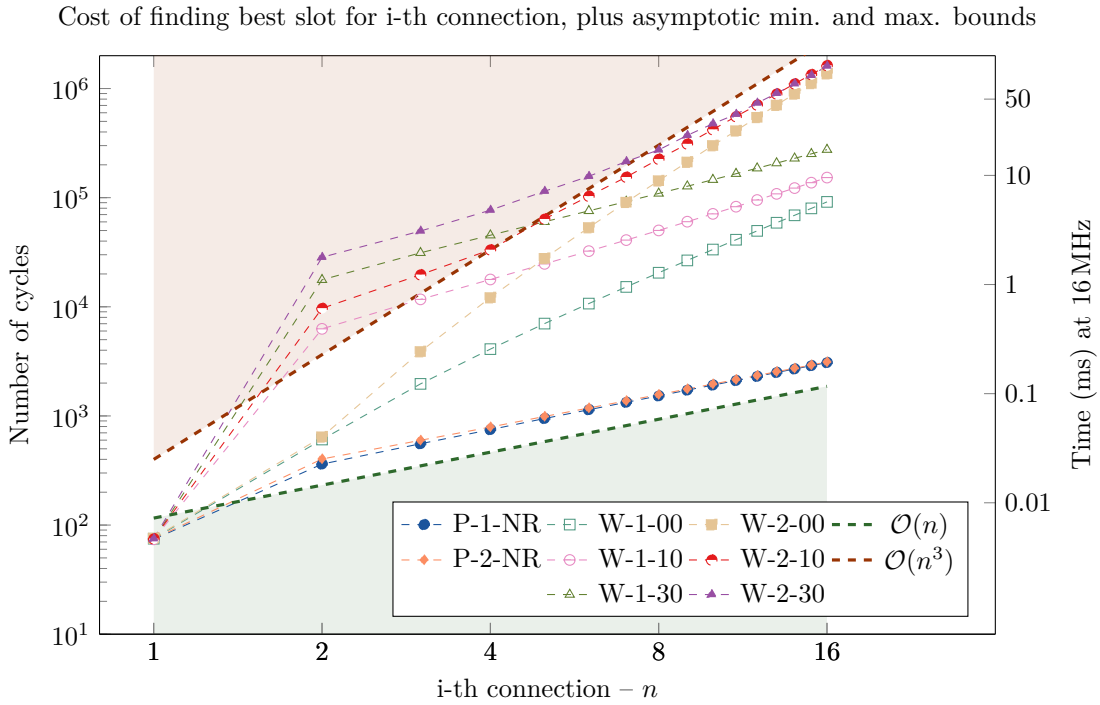


FIGURE 5.18: Benchmark of the slot finder algorithm in various extreme cases in $\log x - \log y$ scale.

5.5 ENHANCED WI-FI DETECTION

SECTION 4.3 proposed a contribution to the current CA algorithm. This enhancement is designed to detect local Wi-Fi signals and identify the strongest Wi-Fi channel among them. Analysing the performance of this improvement first needs a model for the Wi-Fi signals (SECTION 5.5.1) and to define the smoothing parameter, k, to use within the filter (SECTION 5.5.2). Once it is achieved, one can further benchmark the algorithm in various situations (SECTION 5.5.3).

5.5.1 Modeling Wi-Fi signals

Within the ISM band, Wi-Fi can occupy 14 different channels. The width of those channels is adjustable and can vary from 20 MHz to 40 MHz. As a result, the following model can adapt the width of its channels.

In previous Wi-Fi activity measurements (SECTION 5.3), one could observe that the signal's intensity can range from 0.2 V to 1.7 V. The SNR is therefore not constant and varies with the power of emitted Wi-Fi. If

multiple channels are activated, detecting the dominant one also depends on the power ratio between all the channels. Indeed, identifying the maximum between 1.7 V and 0.2 V is easier than between 1.7 V and 1.6 V. Distinguishing channels with similar strength becomes challenging as the noise on the measures increases. Here, this noise is mainly due to the probability of detecting a Wi-Fi signal, which will be referred to as p.

Therefore, this document proposes a second metric: the Signal to Second Biggest Ratio (SSBR). When more than one channel is active, the latter will indicate how distinct the strongest signal is from others. The larger SSBR, the easier the identification. *Note: in the following FIGURES, the SNR and SSBR are NOT expressed in* dB*, but only as a ratio between two measured RSSI levels (*V*). Here, a linear scale was preferred.*

As modern Wi-Fi products use OFDM frequency modulation, the spectra of generated Wi-Fi channels reflect this modulation throughout their shape. The sequence that defines which channel is active at what time is random, but it follows a set of rules:

- **R1:** if a channel is active at time n, then it has a low probability p of being deactivated at time $n+1$;
- **R2:** if a channel is inactive at time n, then it has a low probability p^{n_c} of being activated at time $n+1$, where n_c is the number of current active channels;
- **R3:** the number of simultaneous active channels is capped at a parametrisable value (possibly infinite).

The actual *activation/deactivation* probabilities are not relevant here. The duration of tests is such that every possible case is observed enough times to be statistically significant. The reason to use such rules is to keep better control on the number of active channels.

In addition, the sequence of active channels only updates once in a while, *i.e.*, once every n time indices. This update frequency is also a parameter of the model. A time unit (or index) corresponds to the time between two RSSI scans. Indeed, it is not relevant to observe what happens between two consecutive measurements as the BLE device will never capture it.

5.5.2 Determining the best parameters

One part of the improved CA includes a filter that is parametrised with a smoothing parameter called k. A series of simulations was run to determine to most suited value. Those tests performed a multi-dimensional grid scan, *i.e.*, trying all possible combinations, over to following parameters:

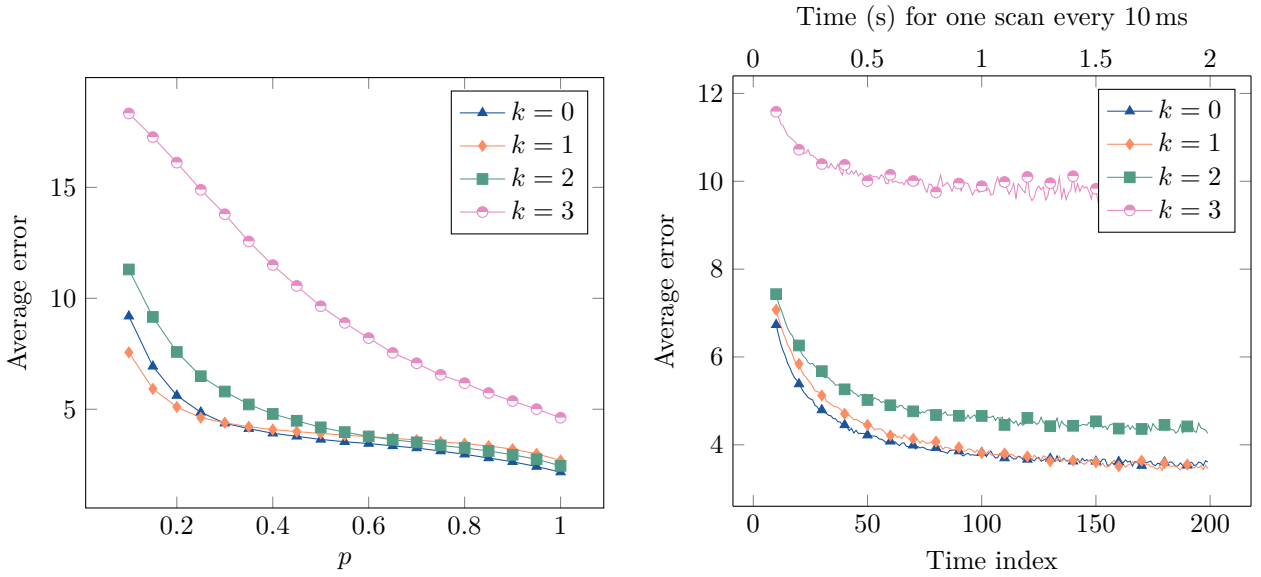
- p: from 0.1 (10 %) to 1 (100 %);
- k: from 0 to 3;
- *Wi-Fi update period*: from 10 (100 ms) to 200 time indices (2 s)
- *SNR*: from 4 to 20;
- *SSBR*: from 1 to 5;
- *Channel width*: 20 MHz and 40 MHz.

The metric of interest is the average error, computed as the absolute difference between the actual channel index and the estimated index. The error is therefore bounded between 0 (estimate is correct) and 37 (worst estimate). Here, the goal is to indicate how far the forecast is from the right answer, similar to a distance. An error of 3, even though it is not perfect, is far better than an error of 20.

As two-dimensional plots cannot display all the measured information, the error is averaged over all the parameters not present on the charts. It is essential to stress that the absolute value of the error is not relevant here but rather the comparison.

First, FIGURES 5.19 and 5.20 both compare the impact of p and the update frequency, for two different channel widths. In the two situations, $k = 0$ and $k = 1$ appear with far better results than the other values of k. At low detection probabilities, the channel identification seems to benefit from the slight smoothing effect of $k = 1$. As one could expect, increasing p reduces the average error. However, the probability of detecting Wi-Fi is not something that one can control, so the choice of k should be equally good, no matter the value of p. On the other hand, increasing the update period also reduces the error, but the analyse is more academic

than practical. It is not common to have a Wi-Fi that updates its channels every 1 or even 2 ms. Of course, $k = 0$ performs the best with high update frequencies, as the response time is shorter, but such rare situations are not a concern for this thesis.

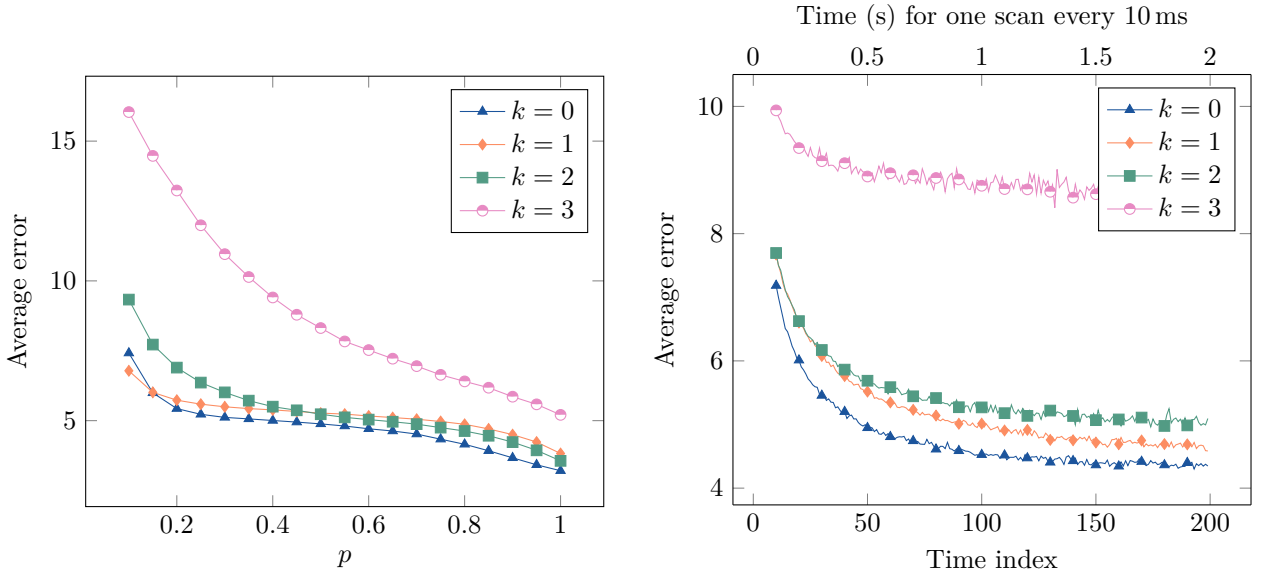


(A) Probability to detect Wi-Fi

(B) Time between two Wi-Fi channels updates

FIGURE 5.19: Average error on Wi-Fi detection with 20 MHz Wi-Fi channels.

On wider Wi-Fi channels, the parameter $k = 0$ performs even better when compared to the others. Part of this result can be explained by the fact that a more extensive spectrum compensates more for the probability effect. As the width grows, the odds of measuring one signal increase as more BLE channels will coexist with the Wi-Fi's. The peak will also be less distinguishable as nearby channels have similar emitting power (due to the almost flat spectrum created by OFDM), and filtering may not help.

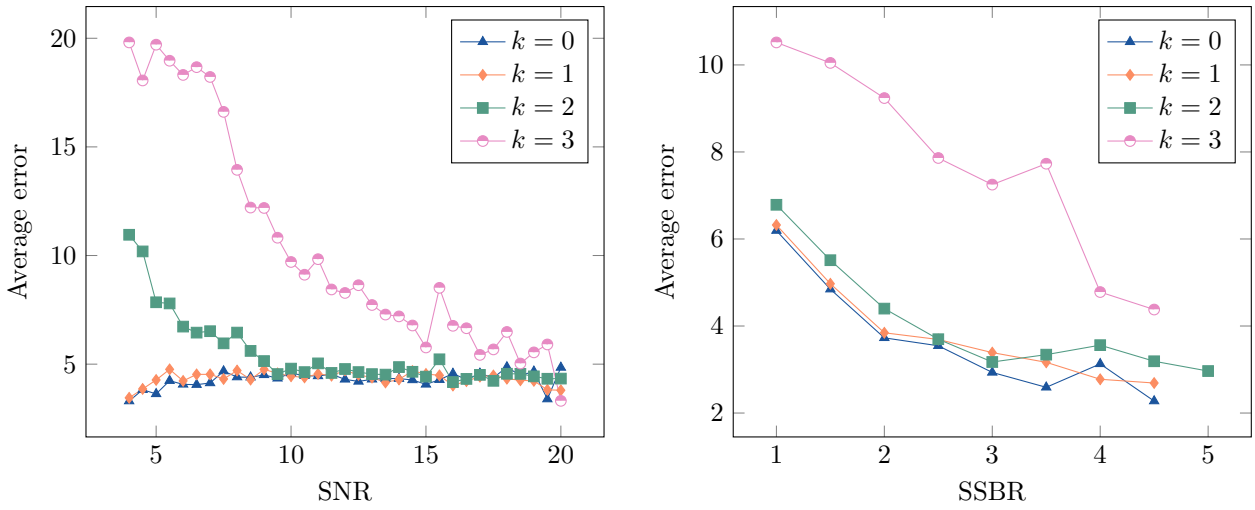


(A) Probability to detect Wi-Fi

(B) Time between two Wi-Fi channels updates

FIGURE 5.20: Average error on Wi-Fi detection with 40 MHz Wi-Fi channels.

FIGURE 5.21 describes the evolution of the error as a function of the Wi-Fi power. Again, an increase in either SNR or SSBR allows for better channel identification. The latter is the consequence of the shorter time needed to distinguish the strongest channel among all the active ones. Indeed, as the convolution combines five different RSSI measurements for each channel, it may take a while before the full convolution has completed. However, when the SNR and the SSBR are high, a partial computation of the convolution can already help identify the principal Wi-Fi signal. Again, one cannot determine those parameters in advance and should therefore choose k values that perform equally well on the whole power spectrum. In this situation, the best values are also $k = 0$ and $k = 1$.



(A) Signal to Noise Ratio (B) Signal to Second Biggest Ratio

FIGURE 5.21: Average error on Wi-Fi detection.

Finally, the different simulations indicate that both $k = 0$ and $k = 1$ outperform the other smoothing parameter values. The exponential trend from the filter is evident as the error ratio between $k = 3$ and $k = 2$ is much larger than the ratio between $k = 2$ and $k = 1$. As a result, the following simulations only consider the two best values for k.

5.5.3 Other benchmarks

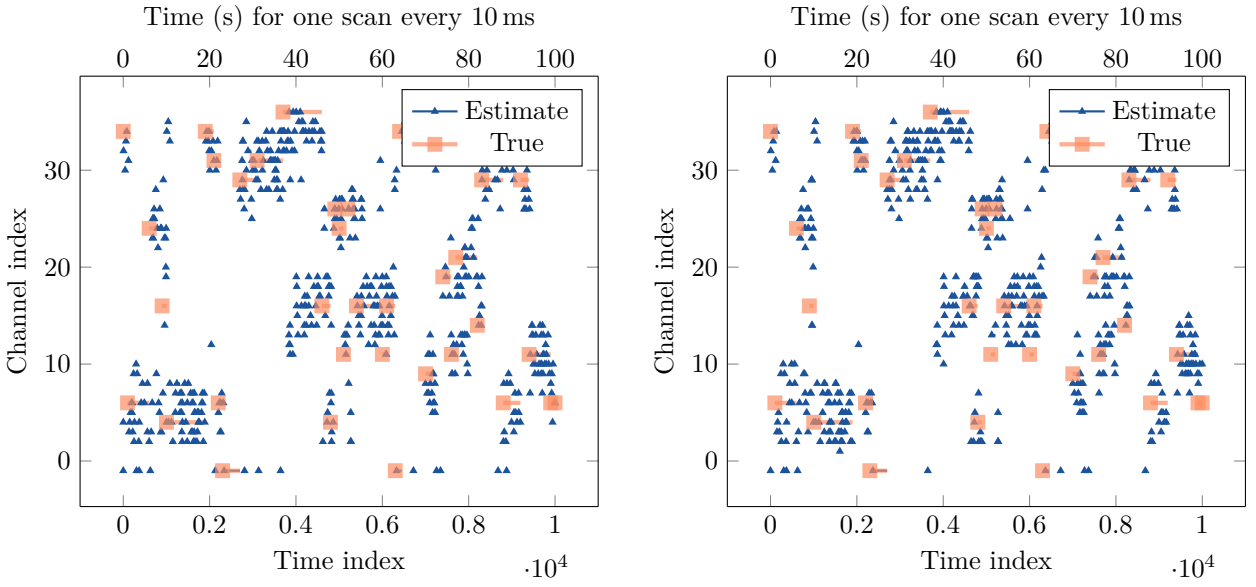
A final series of benchmarks consists of observing the time evolution of the channel estimate and seeing how quickly it responds to changes. The Wi-Fi channels are 20 MHz-wide, and an update is triggered every hundred time units (1 s). As a reminder, an update in the sequence does not necessarily reflect changes in the active channels. The probability of detection, p, and the maximum number of Wi-Fi signals, $n_{c,\max}$, are test parameters mentioned in the caption of the following FIGURES. The following SECTIONS analyse the results of simulations from a very unfavourable case to an ideal situation. For consistency during testing, the numerical simulations use the same random seed to compare the $k = 0$ and $k = 1$ scenarios.

Crowded spectrum, low detection probability

FIGURE 5.22 shows how the algorithm performs in a crowded spectrum with sometimes up to 4 active channels at a time. FIGURES 5.22a and 5.22b plot the estimate channel index against the true value. As the signals do not change instantly, the true index can keep the same value for a few seconds. If the method were perfect, all the blue triangles would lie on the same lines as the orange squares. The bottom line refers to the index -1, meaning that no Wi-Fi signal is active/detected.

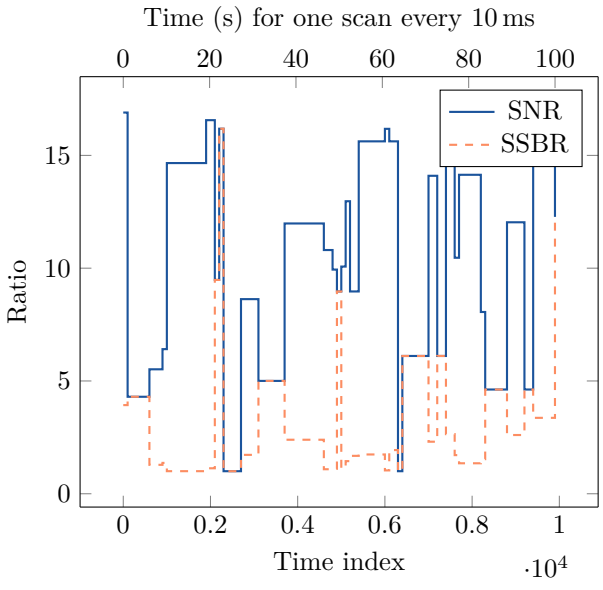
The oscillations around the right channel reflect the average error of ~ 5 channels found in the previous SECTION. Nonetheless, one can already observe that the method can rapidly regroups its estimate around the actual value. Transposed into frequencies, an error of 5 channels corresponds roughly to $5 \cdot 2\,\text{MHz} = 12\,\text{MHz}$.

FIGURES 5.22c and 5.22d both describe the Wi-Fi activity over time in terms of power and number of active channels. As one can observe that, near $t = 20\,\mathrm{s}$, high SNR and SSBR conduct to a smaller variation around the exact channel index. It is reflected by more triangles forming a horizontal line. Here, the SSBR seems to have more impact on those variations than the SNR has.

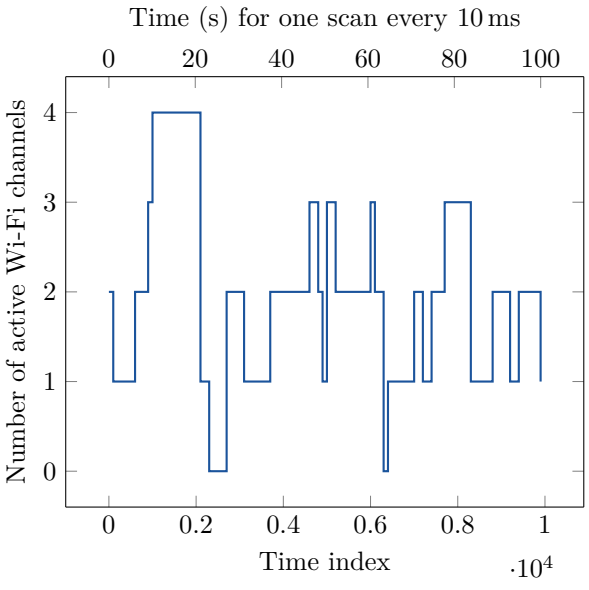


(A) $k = 0$

(B) $k = 1$



(C) Signal ratio



(D) Wi-Fi activity

FIGURE 5.22: Average error on Wi-Fi detection with $p = 30\,\%$ and $n_{c,\max} = \infty$.

One could be tempted to average the estimated channel across time to compensate for the oscillations, but it would hardly work. Indeed, averaging assumes that values oscillate around a more-or-less constant mean. The problem here is that sequence in which the channels are measured, *i.e.*, using FH, is not compatible with an efficient time average. Each channel is only measured once every 37 scans, which means that the time

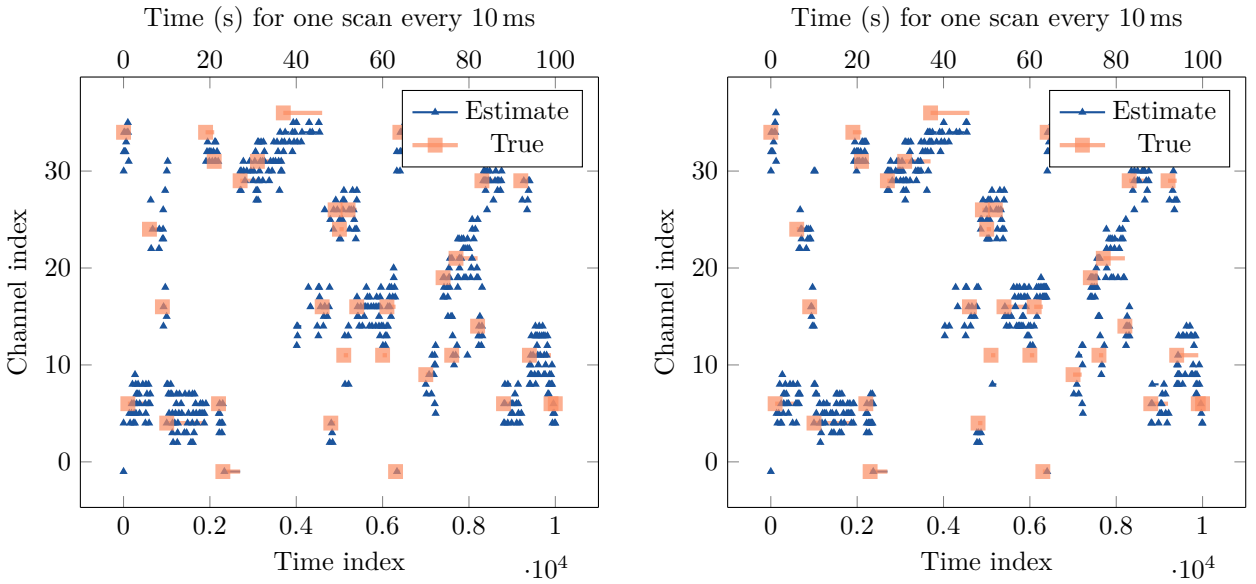
average should reflect this duration by, at least, going through the last 37 estimates. Even though one could quickly build such a moving average, it would dramatically decrease the algorithm's performance in terms of time. Instead, an attempt to use a shorter spatial average, *i.e.*, averaging over three contiguous channels, was performed but resulted poorly.

Finally, the benefits of using a filter, *i.e.*, when $k = 1$, are slight but are there. The filter reduces the number of outliers, and it is more visible in the FIGURES hereunder.

Crowded spectrum, high detection probability

As shown in FIGURE 5.23, the transition from $p = 30\,\%$ to $p = 80\,\%$ greatly improves the precision of the method by regrouping all the estimates closer to the true value.

This precision increase is a result of the RSSI value being also more precise. As the probability of detecting a Wi-Fi signal increases, the RSSI values used for the convolution are getting closer to the reality of the spectrum. Therefore, the peak is more likely to be at the correct position which, thus, decreases the number of outliers.



(A) $k = 0$ (B) $k = 1$

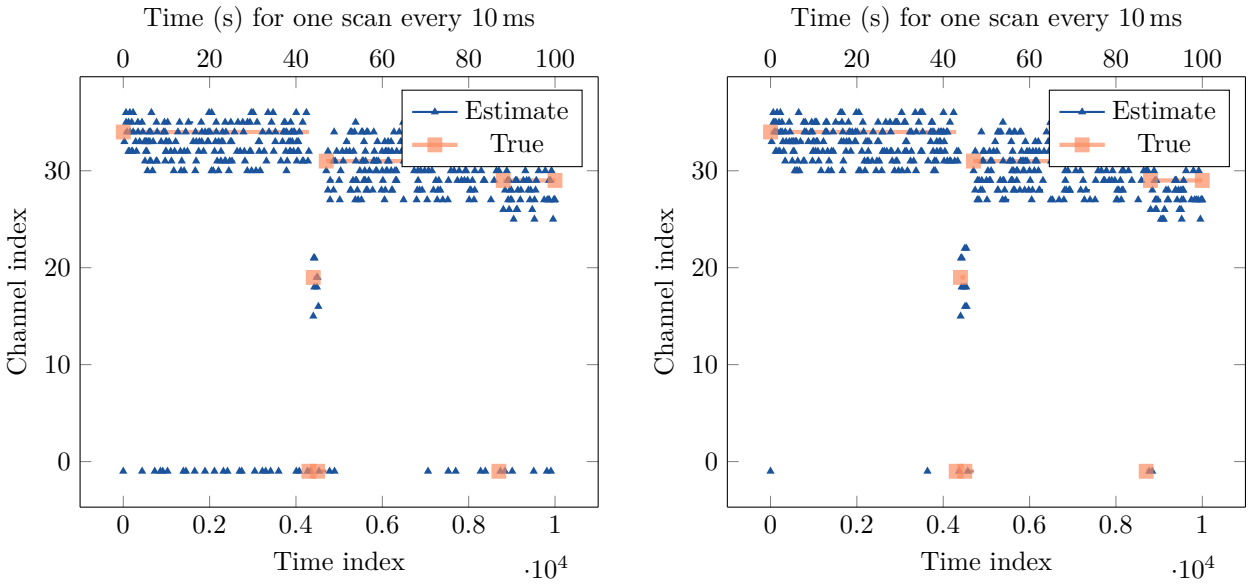
FIGURE 5.23: Average error on Wi-Fi detection with $p = 80\,\%$ and $n_{c,\max} = \infty$.

Uncongested spectrum, low detection probability

Precedent charts presented the performance in a possibly very congested spectrum, *i.e.*, when many Wi-Fi channels are active simultaneously. However, many BLE devices only experience one or two Wi-Fi channels at a time.

This SECTION presents a situation similar to what was found in the garage or in the basement (SECTION 5.3): a low detection probability and maximally one (FIGURE 5.24) or two (FIGURE 5.25) Wi-Fi channels active in parallel.

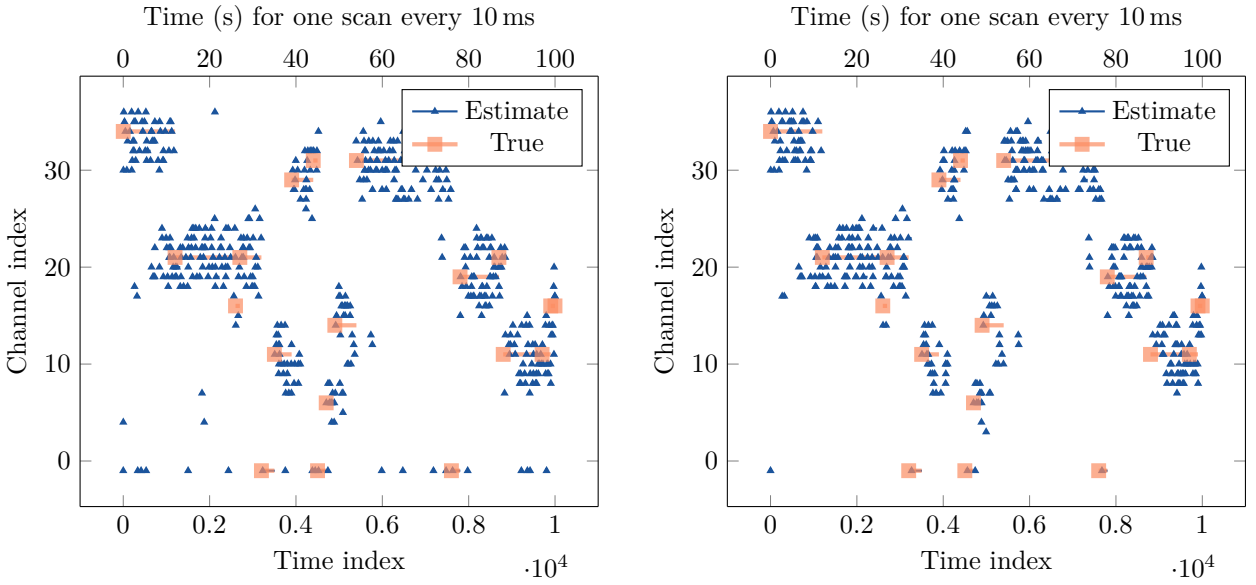
Here, FIGURES 5.24b and 5.25b emphasise the advantage of filtering as they result in almost no false-negatives (index is -1) when compared to situations that do not use any filter (FIGURES 5.24a and 5.25a).



(A) $k = 0$ (B) $k = 1$

FIGURE 5.24: Average error on Wi-Fi detection with $p = 30\,\%$ and $n_{c,\max} = 1$.

For both $n_{c,\max} = 1$ and $n_{c,\max} = 2$, the average error is the same (~ 3 or 4 channels). The rate at which the peak Wi-Fi signal is moving does not seem to deteriorate the quality of the estimate as the algorithm is likely to have low response time. For example, the rapid variations between 40 s and 50 s are very correctly captured.



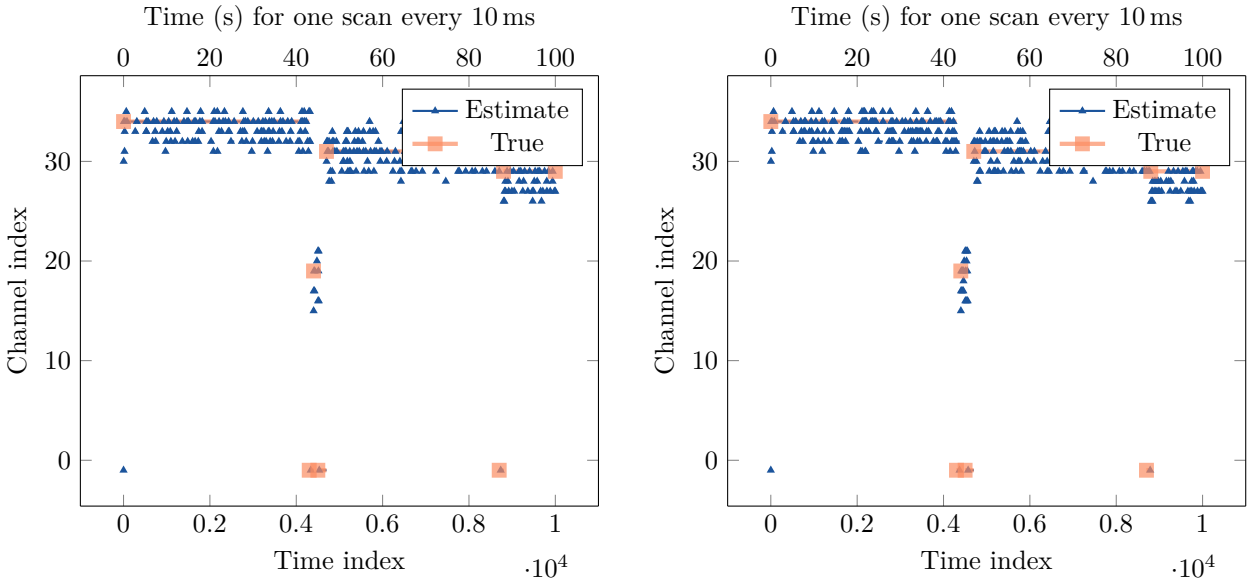
(A) $k = 0$ (B) $k = 1$

FIGURE 5.25: Average error on Wi-Fi detection with $p = 30\,\%$ and $n_{c,\max} = 2$.

Uncongested spectrum, high detection probability

Here, the following plots present a situation similar to what was found in the office (SECTION 5.3): a high detection probability and maximally one (FIGURE 5.26) or two (FIGURE 5.27) Wi-Fi channels active in parallel.

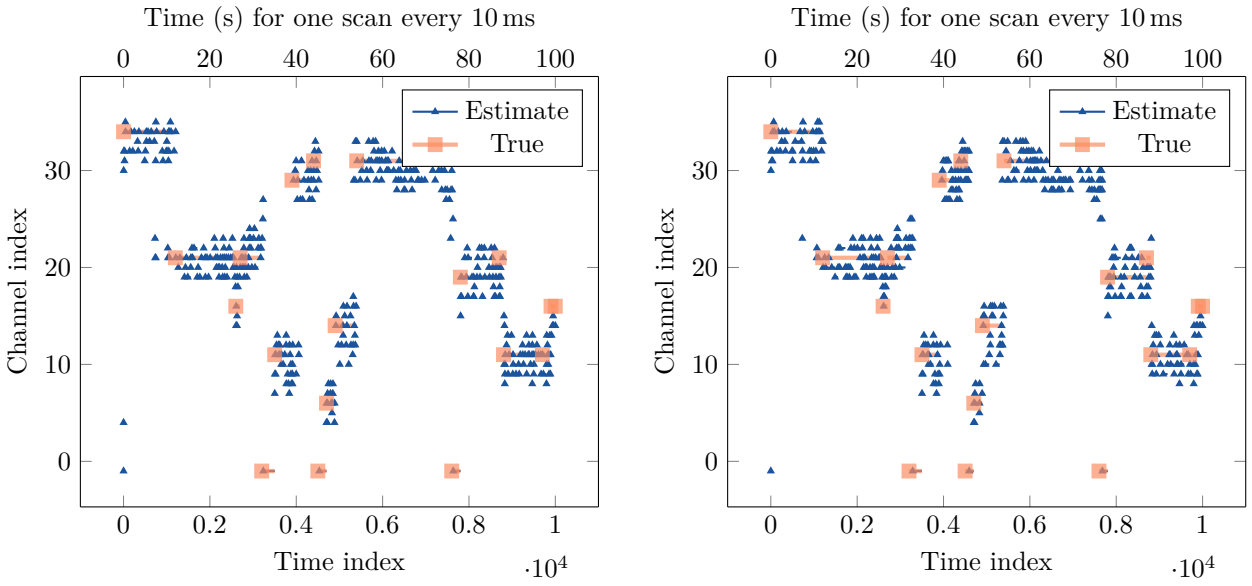
Once more, increasing the p reduces the average error. However, the benefit, when compared to previous situations, is small. This can be explained by FIGURE 5.19a: between 30 % and 80 %, the error curve is almost flat. The interest for higher probabilities is more significant in the 0 % to 30 % and 80 % to 100 % ranges.



(A) $k = 0$ (B) $k = 1$

FIGURE 5.26: Average error on Wi-Fi detection with $p = 80\,\%$ and $n_{c,\max} = 1$.

For $p = 80\,\%$, outliers are remarkably less present. As such, filtering the RSSI measures is less of a need. Left and right situations almost have the same results.

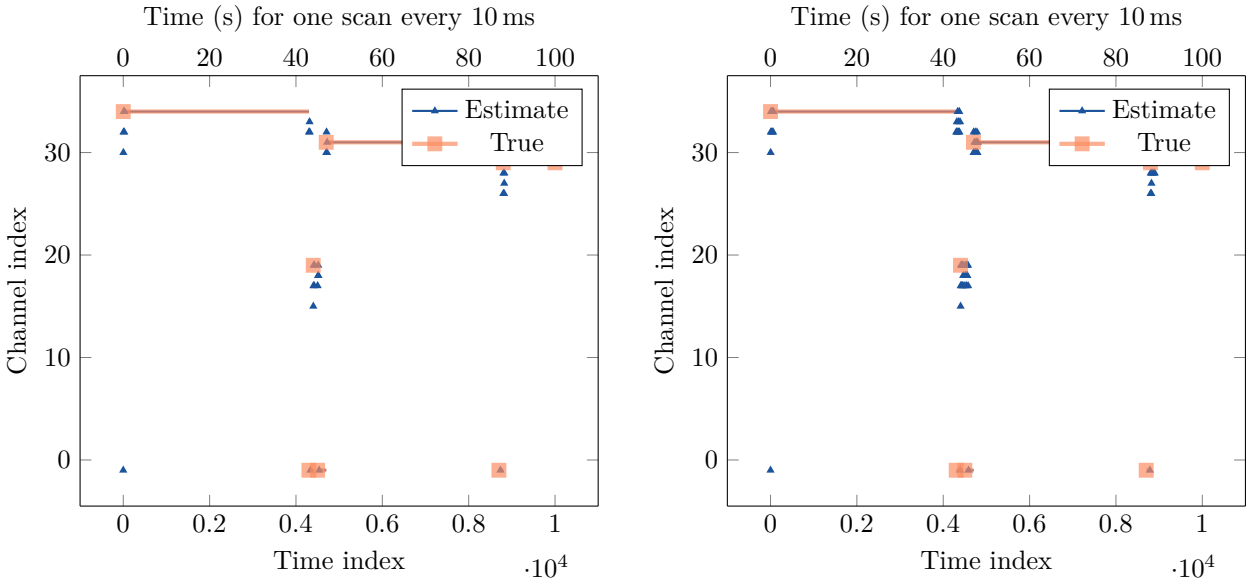


(A) $k = 0$ (B) $k = 1$

FIGURE 5.27: Average error on Wi-Fi detection with $p = 80\,\%$ and $n_{c,\max} = 2$.

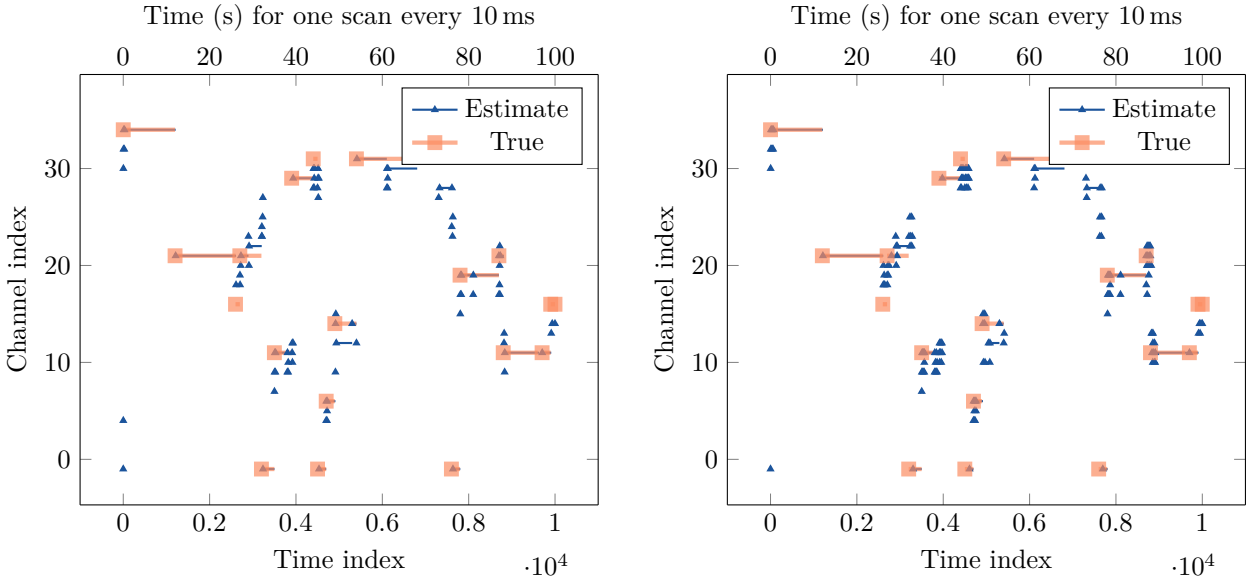
Uncongested spectrum, perfect detection probability

Lastly, FIGURES 5.28 and 5.29 show that, in the case of 100 % probability of detecting a Wi-Fi signal, the channel index error gets very close to zero. Such situations are probably rarely encountered, but those simulations were run as a proof of concept. Those results are essential as they validate that the algorithm will tend towards the best solution. Again, the smoothing parameter has no much importance here as the number of outliers is near zero. The only present outliers are the ones on the left part of each plot. This part of the graphics is not relevant since it is highly dependent on the system's initial conditions. Every channel's RSSI starts at zero, so, understandably, the first detection takes more time than the others.



(A) $k = 0$ (B) $k = 1$

FIGURE 5.28: Average error on Wi-Fi detection with $p = 100\,\%$ and $n_{c,\max} = 1$.



(A) $k = 0$ (B) $k = 1$

FIGURE 5.29: Average error on Wi-Fi detection with $p = 100\,\%$ and $n_{c,\max} = 2$.

6 Conclusion

BLE technology will undoubtedly evolve further over the next few years. The range of applications for BLE technology is growing by the day, bringing many design challenges with it.

After a brief introduction on how BLE works, Chapter 3 discussed the different challenges currently faced and how to deal with them. Solving these problems requires knowledge of both computer science and wireless communications. As it has been seen, the interest in BLE is mainly related to its low power consumption, which also adds constraints in terms of the hardware that can be used. Several of these challenges have also been further analysed, highlighting the various vital points. For most of the parameters defining BLE connections, there are many values that these can take. Therefore, this Chapter also detailed several values that are typical for modern industry.

Next, Chapter 4 addressed two of the many challenges presented: communication task scheduling and coexistence with Wi-Fi. Concerning the first one, a new method was devised based on counting the number of collisions. After several tests conducted in Chapter 5, this algorithm proved to be promising for its performance. Indeed, this algorithm makes it possible to find an excellent solution to the problem in less than 100 ms. Moreover, under some reasonably realistic assumptions, it is possible to reduce this time to 10 ms, or even 1 ms in the best case. However, it is important to stress that this last improvement requires implementing a pre-computation that strongly depends on the conditions in which the BLE product is located.

The second contribution concerned the improvement of the CA algorithm. The latter was aimed at enabling any BLE product to detect the strongest of the surrounding Wi-Fi signals. In order to meet the strict time constraints, the enhancement was developed in such a way that its execution would not add more than 20 µs to the CA (without taking into account the RSSI measurement). This addition to the algorithm consists of three blocks: an IIR filter, a partial convolution and a threshold filter. As discussed in the simulations, the filter can have both a beneficial and adverse effect but has proven to be robust to outliers. With this advancement, the BLE product is now able to quickly (within tens of milliseconds) detect the position of the Wi-Fi channel with a minimum accuracy of between 2 MHz and 12 MHz, depending on the case.

In partnership with an industry working on state-of-the-art BLE products, this thesis allowed to link theoretical concepts to practical applications encountered in everyday life. This paper aimed to provide a good understanding of modern problems while attempting to contribute to their resolution. The results obtained from these two contributions are encouraging and may well be incorporated into future BLE products. A test phase on real hardware still needs to be performed, but unfortunately, it could not be conducted during this thesis. In future work, it would be interesting to optimise the different contributions according to the end product. Indeed, the tests carried out so far were very generic, but it might be more appropriate to adjust the parameters to more specific situations to further improve performance.

Finally, the work accomplished will hopefully be a small stone in the building of BLE. This technology is the result of many years of innovation and evolution. Working on technology with so many practical applications and access to the general public is a chance to see one's work have an impact in the near future.

Bibliography

[1] R. Heydon, *Bluetooth Low Energy, the developer's handbook*. Reading, Massachusetts: Prentice Hall, 2013.

[2] Ray, "Evolution of WiFi then and now," https://www.ray.life/evolution-of-wifi-then-and-now/, 2020.

[3] Z. Sherali, S. Farhan, and B. Zubair, "25 years of Bluetooth technology," https://www.mdpi.com/1999-5903/11/9/194/pdf, Sep 2019.

[4] Bestwireless, "What is the difference between Bluetooth and Bluetooth Low Energy?" https://bestwirelessbluetoothheadphones.com/wiki/bluetooth-low-energy, Jan 2020.

[5] Bluetooth SIG, "The expansion of the connected device market," https://www.bluetooth.com/wp-content/uploads/2020/03/2020_Market_Update-EN.pdf.

[6] Q. D. La, D. Nguyen-Nam, M. v. Ngo, H. Hoang, and T. Q. Quek, "Dense deployment of BLE-based body area networks: A coexistence study," *IEEE Transactions on Green Communications and Networking*, vol. PP, pp. 1–1, Jul 2018.

[7] U. Wetzker, I. Splitt, M. Zimmerling, K. Römer, and C. A. Boano, "Troubleshooting wireless coexistence problems in the industrial Internet of Things," Aug 2016.

[8] L. Januszkiewicz, "Analysis of human body shadowing effect on wireless sensor networks operating in the 2.4 GHz band," *Sensors*, vol. 18, no. 10, p. 3412, Oct 2018. [Online]. Available: http://dx.doi.org/10.3390/s18103412

[9] Core Specification Working Group, *Overview of Bluetooth Core Specification v5.2*. Bluetooth SIG Group, 2020.

[10] Wikipedia contributors, "List of WLAN channels," https://en.wikipedia.org/wiki/List_of_WLAN_channels, Mai 2021.

[11] ——, "Asynchronous Connection-Less," https://en.wikipedia.org/wiki/Asynchronous_Connection-Less, Jun 2021.

[12] A. Mohammad, "Bluetooth 5 speed: How to achieve maximum throughput for your BLE application," https://www.novelbits.io/bluetooth-5-speed-maximum-throughput/, Sep 2017.

[13] S. W. Song, Y. S. Lee, F. Imdad, M. T. Niaz, and H. S. Kim, "Efficient advertiser discovery in Bluetooth Low Energy devices," *Energies*, vol. 12, no. 9, p. 1707, May 2019. [Online]. Available: http://dx.doi.org/10.3390/en12091707

[14] S. Rajesh, G. Anita, V. Shaik, K. Amit, B. Dharam, and K. Prabin, "Forest 4.0: Digitalization of forest using the Internet of Things (IoT)," *Journal of King Saud University - Computer and Information Sciences*, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157821000483

[15] *Lithium Button Cell*, https://www.mega-piles.com/docs/pdf/461.pdf, ANSMANN, Oct 2019, no. 704080.

[16] Aislelabs, "The hitchhikers guide to iBeacon hardware: A comprehensive report by Aislelabs," https://www.aislelabs.com/reports/beacon-guide/, 2015.

[17] Core Specification Working Group, *Bluetooth Core Specification v5.2*. Bluetooth SIG Group, 2019.

[18] Silicon Labs, "TX power limitations for regulatory compliance (ETSI, FCC)," https://docs.silabs.com/bluetooth/latest/general/system-and-performance/tx-power-limitations-for-regulatory-compliance-etsi-fcc.

[19] E. Tsimbalo, X. Fafoutis, and R. Piechocki, "Fix it, don't bin it! - CRC error correction in Bluetooth Low Energy," Dec 2015.

[20] Z. Hui, N. Charles, K. Thomas, and S. Howard, "Frequency accuracy & stability dependencies of crystal oscillators," 2008.

[21] TestWorld, "Understanding frequency accuracy in crystal controlled instruments," https://www.testworld.com/wp-content/uploads/understanding-frequency-accuracy-in-crystal-controlled-instruments.pdf.

[22] IEEE-UFFC, *Introduction to Quartz Frequency Standards Static Frequency versus Temperature Stability*. IEEE.

[23] BDTIC, "LTC1799 - 1 khz to 33 MHz resistor set SOT-23 oscillator," http://www.bdtic.com/en/linear/LTC1799.

[24] All About Circuits, "Choosing the right oscillator for your microcontroller," https://www.allaboutcircuits.com/technical-articles/choosing-the-right-oscillator-for-your-microcontroller/, May 2016.

[25] D. Coleman, "What is a clear channel assessment (cca)?" https://www.extremenetworks.com/extreme-networks-blog/what-is-a-clear-channel-assessment-cca/, Mar 2021.

[26] S. Grimaldi, A. Mahmood, and M. Gidlund, "Real-time interference identification via supervised learning: Embedding coexistence awareness in iot devices," *IEEE Access*, vol. 7, pp. 835–850, 2019.

[27] J. Metcalf, "16-bit xorshift pseudorandom numbers in Z80 Assembly," http://www.retroprogramming.com/2017/07/xorshift-pseudorandom-numbers-in-z80.html, Jul 2017.

[28] F. Steve, *Intel x86 JUMP quick reference*, http://unixwiz.net/techtips/x86-jumps.html, Unixwiz.

[29] J. Jobin, "Branchless programming. does it really matter?" https://dev.to/jobinrjohnson/branchless-programming-does-it-really-matter-20j4, Feb 2021.

[30] N. Shafiei, "Solving linear recurrence relations," http://nms.lu.lv/wp-content/uploads/2016/04/21-linear-recurrences.pdf, Apr 2016.

List of Figures

List of Tables

List of Source Codes

Appendices

Programming

A.1 Branching concept and branchless programming

Branching is a programming concept that describes the fact that, during program execution, the processor may need to choose between multiple paths, *i.e.*, multiple branches. These branches are very common as they are introduced by control flow statements: `if`, `else`, `for`, `while`, etc.

A fundamental example code with branching is shown in Listing 8: a number is read from the standard input and, depending on the value that is now stored into the variable `a`, the program will either return `0` or `1`.

```
1  #include <stdio.h>
2
3  int main() {
4      int a;
5      scanf("%d", &a);
6      if (a > 0)
7          return 0;
8      else
9          return 1;
10 }
```

Listing 8: Example of code with branching.

In Listing 8, the branching happens when executing line 6, *i.e.*, `if (a > 0)`. If the inequality is satisfied, then the execution continues on line 7 and returns. Otherwise, the program jumps to line 9 and returns. In Assembly languages, these jumping instructions are often prefixed by a `b` for branch or a `j` for jump. In GCC x64-64, there are a variety of branching instructions: `jmp`, `je`, `jne`, `jle`, and many more [28]. While `jmp` is the classical branching operation, all other branching are conditional, *i.e.*, they only execute if a specific condition is met.

In the example studied, the comparison `a > 0` will result in a `test` instruction. The latter is equivalent to subtracting `1` from `a` and setting the appropriate value to the following flags: `OF` (`1` if overflow), `ZF` (`1` if result is zero) and `SF` (`1` if result is negative).

Waiting for the condition to be verified can be problematic with processors using a pipelined architecture. Indeed, the benefit of pipelining instructions can deteriorate when instructions depend on the result of a previous instruction. This is exactly what happens when `jle` needs `test` to complete before determining if the

branch `.L2:` will be taken or not. Instead of waiting and doing nothing for some clock cycles, the processor often guesses the branch that will be taken, continues on this branch, and, if it was the wrong branch, will go back to the correct one as soon as `test` terminated. Using the marvellous online Compiler Explorer tool, the Assembly output has been generated using GCC for an x86-64 architecture and is shown in Listing 9. The branching principle can be applied to all programming languages, compilers, and architectures, but the resulting Assembly code will most likely be different to the one presented here.

```
.LC0:
        .string "%d"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        lea     rax, [rbp-4]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    __isoc99_scanf
        mov     eax, DWORD PTR [rbp-4]
        test    eax, eax
        jle     .L2
        mov     eax, 0
        jmp     .L4
.L2:
        mov     eax, 1
.L4:
        leave
        ret
```

Listing 9: GCC x84-64 Assembly of code with branching.

However, branch guessing is not perfect and can deteriorate the performance of programs. Relying on guesses is why programmers may prefer to avoid branching instructions when possible to gain performance. This mindset introduces the field of branchless programming. The goal hidden behind this concept is to replace branching instructions with a set of instructions that give the same results but without any branch, hoping that it will improve performance. One other advantage that branchless programming has is creating deterministic time applications, *i.e.*, programs when the exact number of cycles that it will take can be known in advance. Knowing the exact duration of a function can be relevant when applications have stringent time constraints to meet, such as for communication protocols.

For the code presented in Listing 8, a branchless version is proposed in Listing 10. Here, the two `return` statements have been concatenated into one using a logical operator, that directly returns `0` or `1`.

```
#include <stdio.h>

int main() {
    int a;
    scanf("%d", &a);
    return a < 1;
}
```

Listing 10: Branchless version of example code.

This version results in a code with fewer instructions, see Listing 11, and that will hopefully execute faster than the previous code. The absence of branches is denoted by the fact that `.L2:` and `.L4:` are not needed anymore. In most cases, a program will benefit from branchless programming only if the removed branches were placed in a loop or some part of the code repeatedly called many times.

```
.LC0:
        .string "%d"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        lea     rax, [rbp-4]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    __isoc99_scanf
        mov     eax, DWORD PTR [rbp-4]
        test    eax, eax
        setle   al
        movzx   eax, al
        leave
        ret
```

Listing 11: GCC x84-64 Assembly of branchless code.

Last but not least, branchless programming is not always the best performance solution, and it can also quickly deteriorate the readability of the code. As discussed in [29], finding an efficient way to remove branches from a code is not an easy task, and the compiler might be better at doing this than you. This is exactly what is shown in Listing 12: using the optimisation flag `-O3`, the compiler is able to optimise both codes (Listings 8 and 10) to the same results, that contains even less instructions that the manually optimised one.

```
.LC0:
        .string "%d"
main:
        sub     rsp, 24
        xor     eax, eax
        mov     edi, OFFSET FLAT:.LC0
        lea     rsi, [rsp+12]
        call    __isoc99_scanf
        mov     edx, DWORD PTR [rsp+12]
        xor     eax, eax
        test    edx, edx
        setle   al
        add     rsp, 24
        ret
```

Listing 12: GCC x84-64 Assembly when optimisation flag `-O3` is used. The result is the same for both code versions.

A.2 Connection collisions graphical interface

```python
import numpy as np; import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, Button
from math import gcd as compute_gcd

def compute_lcm(x, y):
    lcm = (x*y)//compute_gcd(x,y)
    return lcm

def n_collisions(T0, T1, L0, L1, t1):
    gcd = compute_gcd(T0, T1); lcm = compute_lcm(T0, T1)

    kmax, rmax = divmod(L0 - (t1 % gcd), gcd)
    kmin, rmin = divmod(T0 - L1 - (t1 % gcd), gcd)
    kmax -= rmax == 0
    kmin += 1

    n = lcm // T1
    return (kmax + 1)*(kmax > -1) + (n - kmin)*(kmin < n)

init_T0 = 80; init_T1 = 70; init_L0 = 20; init_L1 = 30; init_t1 = 40
t = np.linspace(0, 700, 701); sliders = {}

def y(t, t0, T, L):
    rem = t % T
    return (rem >= t0) & (rem <= t0 + L)

fig, ax = plt.subplots()
fig.set_size_inches(w=14/2.54, h=10/2.54)

left_vline = ax.axvline(0, color='r')
right_vline = ax.axvline(0, color='r')

text_lcm = fig.text(.2, .95, ""); text_m = fig.text(.4, .95, "")
text_n = fig.text(.6, .95, ""); text_coll = fig.text(.8, .95, "")

def update(event=None, init=False):
    t0 = 0
    T0 = sliders["$T_0$"].val; T1 = sliders["$T_1$"].val
    L0 = sliders["$L_0$"].val; L1 = sliders["$L_1$"].val
    left = t1 = sliders["$t_1$"].val
    gcd = compute_gcd(T0, T1); lcm = compute_lcm(T0, T1)
    right = t1 + compute_lcm(T0, T1)
    left_vline.set_xdata([left, left])
    right_vline.set_xdata([right, right])

    m = lcm // T0; n = lcm // T1
    n_coll = n_collisions(T0, T1, L0, L1, t1)

    text_lcm.set_text("LCM: %6ds" % lcm); text_m.set_text(f" = %3d x $T_0$" % m)
    text_n.set_text(f" = %3d x $T_1$" % n);
    text_coll.set_text(f"Collisions: %3d" % (n_coll,))
```

```
    y0 = y(t, t0, T0, L0); y1 = y(t, t1, T1, L1)

    if init:
        ax.set_xlim(np.min(t), np.max(t))
        ax.set_ylim(0, 1)

    ax.collections.clear()
    a = ax.fill_between(t, y0, color="#315e99", alpha=.5, step="post")
    b = ax.fill_between(t, y1, color="#fe9461", alpha=.5, step="post")
    ax.legend([a, b], ["Conn. 0", "Conn. 1"], loc="upper right")

    fig.canvas.draw_idle()

ax.set_xlabel('Time (s)')

axcolor = 'lightgoldenrodyellow'

sliders_params = {
    "$T_0$": (10, 100, init_T0),
    "$T_1$": (10, 100, init_T1),
    "$L_0$": ( 1, 40, init_L0),
    "$L_1$": ( 1, 40, init_L1),
    "$t_1$": ( 5, 50, init_t1),
}

# adjust the main plot to make room for the sliders
plt.subplots_adjust(bottom=0.05 * len(sliders_params) + .15, top=0.9)

bottom = 0.05
for name, (valmin, valmax, valinit) in sliders_params.items():
    _ax = plt.axes([0.1, bottom, .8, 0.03], facecolor=axcolor)
    sliders[name] = Slider(ax=_ax, label=name, valmin=valmin, valmax=valmax,
                           valinit=valinit, valstep=1, valfmt="%-ds",
                           color="#315e99")
    sliders[name].on_changed(update)
    bottom += .05

def reset(event):
    for slider in sliders.values():
        display("reset")
        slider.reset()

resetax = plt.axes([0.02, .935, 0.1, 0.04], facecolor=axcolor)
button = Button(resetax, 'Reset', color=axcolor, hovercolor='0.975')
button.on_clicked(reset);

update(init=True)

update()

plt.show()
```

Listing 13: Python code of the connection collisions graphical interface.

Mathematical proofs and identities

B

B.1 Modular arithmetic

Property B.1.1 (Repeated modulo operation with common divisor)**.** *For any integer A, and any non zero integers B and C, computing $(A \bmod BC) \mod C$ is equal to $A \bmod C$. The relation still holds if B and C are permuted.*

Proof. First, A can be decomposed as the product of a quotient q time a dividend d, plus a remainder r. Then, using BC as a first dividend:

$$A = q_1 BC + r_1 \tag{B.1}$$

with r_1 the result of $A \bmod BC$.
In turn, r_1 can also be decomposed using C as a dividend:

$$A = q_1 BC + q_2 C + r_2 \tag{B.2}$$

with r_2 the result of $(A \bmod BC) \mod C$.
But, A can also be directly decomposed with C as the dividend:

$$A = q_3 C + r_3 \tag{B.3}$$

with r_3 the result of $A \bmod C$.
Since $q_1 BC + q_2 C \bmod C = q_3 C \bmod C = 0$, it implies that $r_1 = r_2$ and proves the theorem.

■

B.2 Recurrence relations

Property B.2.1 (Solution to non-homogeneous recurrence)**.** *The equation $u[n+1] = Au[n] + B[n]$, with known initial condition $u[0]$, has solution $u[n] = \frac{(Au[0]+B-u[0])A^n - B}{A-1}$, if $B[n]$ is constant,* i.e., *$B[n] = B$. In the general case were $B[n]$ is not constant, the solution is:*

$$u[n] = A^{n-1}\left(\sum_{i=0}^{n-1} A^{-i} B[i]\right) + u[0]A^n \tag{B.4}$$

Proof. First, the case were $B[n]$ is constant will be proved. The non-homogeneous equation can be converted into homogeneous form as follows:

$$u[n+1] = Au[n] + B \tag{B.5}$$
$$u[n+2] = Au[n+1] + B \tag{B.6}$$
$$\implies u[n+2] - u[n+1] = A\left(u[n+1] - u[n]\right) \tag{B.7}$$

The equation now becomes:

$$u[n+2] = (A+1)\,u[n+1] - Au[n] \tag{B.8}$$

The general solution to this equation is:

$$u[n] = C\lambda_1^n + D\lambda_2^n \tag{B.9}$$

with C and D parameters that can be defined based on two given initial conditions $u[0]$ and $u[1]$. The eigenvalues λ_1 and λ_2 are the roots of the characteristic equation:

$$\lambda^2 = (A+1)\,\lambda - A \tag{B.10}$$

The determinant of this polynomial is simply:

$$\Delta = (A+1)^2 - 4A = (A-1)^2 \tag{B.11}$$

Then, the roots are obtained:

$$\lambda_1 = \frac{A+1+\sqrt{\Delta}}{2} = A \qquad \text{and} \qquad \lambda_2 = \frac{A+1-\sqrt{\Delta}}{2} = 1 \tag{B.12}$$

Therefore, (B.9) reduces to:

$$u[n] = CA^n + D \tag{B.13}$$

Then, only one initial condition is needed given that $u[n] \implies u[n+1]$. Therefore, if $u[0]$ is known:

$$u[0] = C + D \tag{B.14}$$
$$u[1] = CA + D = Au[0] + B \tag{B.15}$$
$$C = \frac{Au[0] + B - u[0]}{A-1} \qquad \text{and} \qquad D = -\frac{B}{A-1} \tag{B.16}$$

Finally, the solution of recurrence equation can be written as:

$$u[n] = \frac{\left(Au[0] + B - u[0]\right)A^n - B}{A-1} \tag{B.17}$$

Using the decomposition technique proposed in [30, Page 24] and similar development as above, one can obtain the general solution for non constant $B[n]$:

$$u[n] = A^{n-1}\left(\sum_{i=0}^{n-1} A^{-i}B[i]\right) + u[0]A^n \tag{B.18}$$

for $n \in \mathbb{Z}_0$.

■

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**