Tobias Renzler, BSc

Increasing the Energy Efficiency and Responsiveness of Bluetooth Low Energy (BLE)

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Telematics

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dr. Carlo Alberto Boano

Institute of Technical Informatics

Second supervisor
Dipl.-Ing. Michael Spörk, BSc

Graz, January 2018

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis dissertation.

Date	Signature

Tobias Renzler, BSc

Increasing the Energy Efficiency and Responsiveness of Bluetooth Low Energy (BLE)

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Telematik

eingereicht an der

Technischen Universität Graz

Betreuer

Ass.Prof. Dr. Carlo Alberto Boano

Institut für Technische Informatik

Zweitbetreuer
Dipl.-Ing. Michael Spörk, BSc

Graz, Januar 2018

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum	Unterschrift

Abstract

The low-power consumption and low-cost of Bluetooth Low Energy (BLE) devices raised the popularity of BLE over the past years. Nowadays, BLE is supported by most consumer electronic devices and it is an integral part in everyday application domains. Because it is used in a wide range of applications, the specific requirements of BLE applications are largely different. Furthermore, requirements may change over time and may be user dependent, having an adverse impact on the energy efficiency and responsiveness of BLE. The community is aware of changing application requirements over time: several runtime adaptation techniques for device discovery and connection management are indeed available in the literature. To date, however, there is no work that targets the influence of user behavior on the device discovery process of BLE.

This thesis focuses on making the device discovery process more energy efficient and responsive. It does so by introducing two novel concepts: Adaptive Advertising and Range Extender. To find the appropriate trade-off and adapt it at runtime, Adaptive Advertising is introduced. The latter adapts the advertising interval of a BLE advertiser at runtime, according to a daily schedule, based on user behavior. To increase the responsiveness of BLE even further, the concept of Range Extender is introduced. The latter informs BLE devices about the presence of other nearby BLE devices by establishing a temporary or permanent connection.

Both Adaptive Advertising and Range Extender are generic such that they can be used in different application scenarios. To outline the benefits of both approaches they are evaluated according to a real world application: the Nuki Smart Lock. The implementation of Adaptive Advertising takes place on the Cypress CY8C4248LQI-BL483 chip. An experimental evaluation shows that the energy consumption and the mean device discovery latency can be decreased by 50%. The Range Extender is implemented on Nordic Semiconductor's nRF52 radio chip using the real-time operating system Zephyr. In both Temporary and Permanent Connection mode, the connection partner could be notified in a mean time of less than two seconds. After a successful notification the connection partner may adapt its advertising interval. This reduces the device discovery time of other potential connection partners, depending on the currently used advertising interval, by more than 95%.

Kurzfassung

Auf Grund seiner Energieeffizienz und geringen Kosten hat Bluetooth Low Energy (BLE) in den letzten Jahren an Popularität gewonnen. Heutzutage sind die meisten elektronischen Verbrauchergeräte mit BLE ausgestattet und BLE ist Bestandteil von vielen alltäglichen Anwendungen geworden. Auf Grund des breiten Einsatzgebietes variieren die Anwendungsanforderungen. Zudem sind Anforderungen zeit- und benutzerabhängig. Dies kann einen negativen Einfluss auf die Energieeffizienz und die Reaktionsgeschwindigkeit von BLE haben. Die Forschung ist sich über den zeitlichen Einfluss bewusst. Deshalb gibt es für die Geräteerkennung und das Verbindungsmanagement einige Adaptionstechniken, die zur Laufzeit eingreifen. Allerdings gibt es gegenwärtig keine Adaptionstechniken, die auch den Einfluss des Benutzerverhaltens auf die Geräteerkennung von BLE miteinbeziehen.

Diese Masterarbeit beschäftigt sich mit der Energieeffizienz und der Reaktionsgeschwindigkeit der BLE-Geräteerkennung. Dazu werden zwei neue Konzepte eingeführt: Adaptive Advertising und Range Extender. Adaptive Advertising findet einen geeigneten, zur Laufzeit anpassbaren, Kompromiss. Basierend auf einem täglichen, vom Nutzerverhalten abhängigen, Zeitplan, adaptiert Adaptive Advertising das Advertising Interval eines BLE Advertisers zur Laufzeit. Um die Reaktionsgeschwindigkeit von BLE weiter zu steigern, wird das Konzept des Range Extenders eingeführt. Dieser informiert BLE Geräte mittels einer temporären oder permanenten Verbindung über die Anwesenheit anderer, sich in der Nähe befindenden, BLE Geräte.

Beide Konzepte sind generisch implementiert, so dass ein breites Anwendunsspektrum möglich ist. Um die Vorteile beider Konzepte hervorzuheben, erfolgt die Evaluierung in einem praxisorientierten Szenario, dem Nuki Smart Lock. Die Implementierung von Adaptive Advertising wird am Cypress Chip CY8C4248LQI-BL483 durchgeführt. Die experimentelle Evaluierung zeigt, dass sowohl der Energieverbrauch als auch die mittlere Zeit der Geräteerkennung um 50% verringert werden können. Der Range Extender wird unter Verwendung des Echtzeitbetriebssystems Zephyr am nRF52 Chip von Nordic Semiconductor implementiert. In beiden Modi, Temporary Connection und Permanent Connection, kann das verbundene Gerät innerhalb einer durchschnittlichen Zeit von unter zwei Sekunden verständigt werden. Sobald eine Benachrichtigung erhalten wurde, kann das verbundene Gerät sein Advertising Interval anpassen um die Zeit der BLE-Geräteerkennung für andere potentielle Verbindungspartner zu verkürzen. Abhängig vom aktuellen Advertising Interval, kann diese Reduktion mehr als 95% betragen.

Acknowledgements

This Master's thesis was written during the year 2017 and the beginning of 2018 at the Institute of Technical Informatics at Graz University of Technology.

I am particularly grateful for the assistance given by my supervisor, Carlo Alberto Boano. His encouragement, patience, and constructive critiques led me through this thesis. Furthermore, I would like to offer my special thanks to my second supervisor Michael Spörk for his valuable feedback, his suggestions during the planning and the implementation of this work, and his motivating words whenever needed.

In addition, I thank the company Nuki Home Solutions. Their flexibility and openness made this thesis possible.

I would like to thank my parents for their support and for their guidance through my studies and my life. Moreover, I wish to thank my friends for setting at rest my mind and providing me with new energy. A special thanks to my girlfriend Verena for bringing a smile to my lips even in tough times.

Graz, January 2018 Tobias Renzler

Danksagung

Diese Diplomarbeit wurde im Jahr 2017 und zu Beginn des Jahres 2018 am Institut für Technische Informatik an der Technischen Universität Graz durchgeführt.

Vor allem danke ich meinem Betreuer Carlo Alberto Boano für seine Unterstützung. Seine Ermunterung, seine Geduld und seine konstruktive Kritik haben mich durch diese Arbeit geführt. Darüber hinaus möchte ich mich bei meinem Zweitbetreuer Michael Spörk für sein wertvolles Feedback, seine Empfehlungen während der Planung und der Implementierung der Arbeit und seine motivierenden Worte bedanken.

Zudem bedanke ich mich bei der Firma Nuki Home Solutions. Ihre Flexibilität und Offenheit haben diese Arbeit ermöglicht.

Ich möchte mich bei meinen Eltern dafür bedanken, dass sie mich immer unterstützen und mich durch meine Studienzeit und mein Leben führen. Zudem danke ich meinen Freunden, die mir die nötige Ablenkung ermöglichen und mir damit neue Kraft geben. Ein besonderes Dankeschön geht an meine Freundin Verena, die mir selbst in harten Zeiten ein Lächeln auf die Lippen zaubert.

Graz, Januar 2018 Tobias Renzler

Contents

1 Introduction **11**
1.1 Motivation . . . 12
1.2 Contribution . . . 12
1.3 Outline . . . 14

2 Background **15**
2.1 Bluetooth Low Energy (BLE) . . . 15
2.1.1 Physical Layer (PHY) . . . 17
2.1.2 Link Layer (LL) . . . 17
2.1.3 Host Controller Interface (HCI) . . . 24
2.1.4 Logical Link Control and Adaptation Protocol (L2CAP) . . . 24
2.1.5 Security Manager (SM) . . . 24
2.1.6 Attribute Protocol (ATT) . . . 27
2.1.7 Generic Attribute Profile (GATT) . . . 29
2.1.8 Generic Access Profile (GAP) . . . 29
2.1.9 Bluetooth Low Energy 5.0 . . . 29
2.2 Hardware . . . 30
2.2.1 Nuki Smart Lock (NSL) . . . 31
2.2.2 Cypress Development Kit (CDK) . . . 35
2.2.3 nRF52 Development Kit (nDK) . . . 36
2.2.4 Monsoon Solutions Inc. Power Monitor . . . 38
2.2.5 nRF Power Profiler Kit (nRF PPK) . . . 39
2.3 Zephyr Project . . . 40

3 Related Work **42**
3.1 Smart Locks . . . 42
3.2 BLE Modeling . . . 43
3.3 Adapting Parameters at Runtime . . . 44

4 Modeling BLE Device Discovery **45**
4.1 BLE Advertisements . . . 45
4.1.1 Measurements on the Cypress Development Kit (CDK) . . . 46
4.1.2 Mathematical Model . . . 47
4.1.3 Advertising Payload . . . 49
4.1.4 Outcome . . . 51
4.2 Range . . . 51

4.2.1 Range measurement 51
4.2.2 Maximum Time for Device Discovery 52
4.3 BLE Scanning 53
4.3.1 Mathematical Model 53
4.3.2 Optimal Scan Parameters (OSP) 55
4.3.3 Evaluation 56

5 Adaptive Advertisement **60**
5.1 Algorithm 60
5.1.1 Time of Day (TD) 60
5.1.2 User Behavior (UB) 61
5.2 Design 62
5.3 Implementation 64
5.3.1 Advertising Levels 64
5.3.2 K-means 64
5.3.3 Points of Interest (PoI) 65
5.3.4 Wakeup Array 66
5.4 Evaluation 69
5.4.1 Use Case: NSL 69
5.4.2 Simulation Scenarios and Parameter Settings 70
5.4.3 Power Consumption and Device Discovery Latency 81
5.4.4 Outcome 85

6 Range Extender **86**
6.1 Concept 86
6.1.1 Range Extender - Permanent Connection (RE-PC) 88
6.1.2 Range Extender - Temporary Connection (RE-TC) 88
6.2 Use Case: NSL 89
6.3 Design Possibilities 90
6.3.1 Powerful NSL 90
6.3.2 Powerful RE 92
6.3.3 Current System and RE 93
6.4 Implementation 94
6.4.1 Advertising 95
6.4.2 Scanning and Connection Establishment 95
6.4.3 Scan Requests 95
6.4.4 Nearby Device Service (NDS) 96
6.4.5 Modes 98
6.5 Evaluation 100
6.5.1 Measurement Setup 100
6.5.2 Permanent Connection (PC) 101
6.5.3 Temporary Connection (TC) 104
6.5.4 Outcome 108

7 Conclusion **111**

8 Future Work **112**

Bibliography **114**

List of Figures

2.1 BLE stack and its different layers (adapted from [Gup13]). 16
2.2 BLE device discovery. 18
2.3 Structure of BLE advertising packet according to BLE v4.2. 19
2.4 BLE connection establishment. 21
2.5 Overview of several BLE connection events. 22
2.6 Different Nuki devices [Gmb17]. 31
2.7 Advertising packet structure of the NSL. 32
2.8 Structure of the scan response packet of the NSL. 33
2.9 Cypress' CY8CKIT-042-BLE development kit. 35
2.10 Nordic Semiconductor's nRF52 Development Kit. 37
2.11 PowerTool provided by Monsoon Solutions Inc. 38
2.12 Nordic Semiconductor's Power Profiler Kit. 40

4.1 Mean power consumption of the CDK for different advertising intervals and being in deep sleep (DS) mode. 47
4.2 Progress of a BLE advertising event for the time T_{ADVE} using all three advertising channels. 48
4.3 Maximum time for device discovery. 52
4.4 Device discovery latency dependent on the selected advertising event duration T_{ADVE}, scan interval T_S, and scan window t_S. 53
4.5 Device discovery latency calculation for non continuous and continuous scanning using a discrete time axis. 56
4.6 Device discovery latency dependent on advertising interval and scan window (a) (b) and the gain of OSP compared to a normal continuous scan window (c). 58

5.1 Advertising schedule over the whole day. 63
5.2 Structure of PoI array. 66
5.3 AA-UB algorithm applied on random time values. 71
5.4 AA-UB algorithm applied on single household time values. 73
5.5 AA-UB algorithm applied on family household time values. 75
5.6 AA-UB algorithm applied on family household time values with increased randomness. 76
5.7 AA-UB algorithm applied on family household time values with different number of clusters. 77
5.8 AA-UB ADV_LEVEL influence. 78

5.9 AA-UB NUMBER_OF_CLUSTER influence. 79
5.10 AA-UB w and f influence. 80
5.11 AA-UB LLEMP and SLLEMP influence. 81
5.12 AA-UB schedules based on different data (random, single household, and family household). 83

6.1 Phases of the unlocking mechanism of the NSL 87
6.2 Sending messages from a smart phone to a paired Nuki Smart Lock (NSL). 89
6.3 Powerful NSL: range extender informing the NSL about approaching devices and their addresses. The NSL performs address resolution in order to uniquely identifying the approaching device and decides if it is relevant. . . 91
6.4 Powerful RE: range extender identifies approaching devices, performs address resolution and informs the NSL only if the approaching device is of relevance for a certain NSL. 92
6.5 Packet structure of a single notification when transmitting one or two different device addresses with an MTU of 23 bytes. 97
6.6 Power consumption of the RE in PC mode during high traffic (a) and in standby (b). 102
6.7 Power consumption of the RE in TC mode during high traffic (green) and in standby (yellow). 106
6.8 Measured average latency of the RE in the two modes, PC and TC, depending on different connection and advertising intervals. 109

List of Tables

2.1 Android's possible settings for all versions with BLE support: advertising interval (T_{ADVI}), scan window (T_s), and scan interval (T_S). 20

2.2 Advertising intervals supported by the NUKI Smart Lock. 33

4.1 Power consumption of the CDK, calculated based on the measured energy consumption over a period of 60 seconds. 46

4.2 Time, power and energy consumption for each transmitted byte. 49

4.3 Reducing the energy consumption by reducing the payload length of an advertising event. 50

4.4 Deviation of the calculated energy consumption compared to the measured energy consumption. 50

4.5 Maximum LOS distance of the CDK for RSSI = -87dBm with different transmission power settings. 51

4.6 Measured mean device discovery latency depending on the selected advertising interval ($\mathrm{T_{ADVI}}$) on the CDK, as well as on the scan interval ($\mathrm{T_S}$) and scan window ($\mathrm{t_S}$) on the nDK. 57

5.1 Energy consumption based on different advertising intervals and savings with respect to the standard advertising interval of 417.5 ms. 61

5.2 Advertising levels and their assigned advertising intervals ($\mathrm{T_{ADVI}}$). 82

5.3 Mean power consumption and device discovery latency of the CDK dependent on the used advertising interval ($\mathrm{T_{ADVI}}$). 82

5.4 Estimated ($P_{AA,est.}$) and measured ($P_{AA,meas.}$) mean power consumption of AA in comparison with the mean power consumption of static advertising ($P_{S,meas.}$). 82

5.5 Influence of AA on the mean device discovery latency in comparison with static advertising for different scenarios (random, single household, and family household). 84

6.1 Supported modes of the RE. 98

6.2 *High traffic*: measured current consumption and calculated power consumption of nRF52 in PC mode over 10 minutes for different connection intervals (connInterval). 102

6.3 *Standby*: measured current consumption and calculated power consumption of nRF52 in PC mode over 10 minutes for different connection intervals (connInterval). The measurement excludes the initial connection establishment. 103

6.4 Average notification latency in PC mode over 100 measurements, using different connection intervals. 104
6.5 *High traffic*: measured current consumption and calculated power consumption of nRF52 in TC mode 110 over 10 minutes for different advertising intervals ($\mathrm{T_{ADVI}}$) on the connection partner. 105
6.6 *Standby*: measured current consumption and calculated power consumption of nRF52 in TC mode over 10 minutes. 105
6.7 Average notification latency in TC mode over 100 measurements, using different advertising intervals on the remote device. 107
6.8 Reduction of the device discovery latency by adapting the advertising interval to the fastest possible advertising interval. 110

Chapter 1

Introduction

Nowadays, more and more devices are connected to each other, creating the Internet of Things (IoT). Those devices are not only powerful devices such as laptop and smart phones, but also smaller, constrained embedded devices equipped with sensors. The IoT has many different application areas such as home automation, smart cities, fitness applications, and health monitoring. In most applications devices collect data and communicate it to nearby devices or over the Internet, transforming them into “smart devices” forming a network that is able to communicate without human intervention. The amount of transferred data for smart objects is usually limited, allowing them to use smaller data rates and achieve a higher energy efficiency. This is an important characteristic, as many IoT devices have to operate for a long time powered by batteries, often cell coin sized batteries. This applies to blood glucose, heart rate, or blood pressure monitors. Also fitness trackers, like fitness bands, require a low energy consumption while not needing high data throughput. To save energy, such devices typically employ a low transmission power (at a price of a reduced communication range) and make use of radio duty cycling, where the radio is switched off or kept in a low-power state most of the time. This behavior of IoT devices introduces a common problem: devices are only discoverable for other devices when they have their radio activated and when they are spatially close to each other. Therefore, it is important to provide an efficient device discovery mechanism. Efficiency can be considered in terms of power consumption and responsiveness.

The requirements of low power consumption are often in contrast to the requirements of responsiveness. On the one hand, users need a short discovery time at the cost of a short battery lifetime, which may make a product unusable. On the other hand, products running for several years on coin cell batteries are not able to provide users with a short latency. An example are smart locks, where the user wants the lock to react as soon as he/she is in close range of the door. Two factors diminishes the customer satisfaction: first, long waiting times in front of the door, which are caused by high latency allowing to save energy. Second, the need of frequent battery replacement, allowing to support a short latency. Therefore, it is crucial to find the right trade-off between energy consumption and responsiveness. This trade-off is application-dependent and thus, it is typically neglected.

1.1 Motivation

The motivation for this thesis is based on the crucial trade-off between the energy consumption and responsiveness (i.e., the latency experienced by the end-user) of the discovery of devices based on Bluetooth Low Energy (BLE). BLE is a popular wireless communication protocol supported by most consumer electronic devices, such as smart phones, laptop, and tablets. BLE is especially designed for low-cost, low-range (up to tens of meters), low-power devices using a reduced data rate of 1 Mbps. It meets most of the requirements of smart devices by offering a highly adjustable protocol. As mentioned, devices operating in the IoT are heavily dependent on the energy consumption.

Using an off-the-shelf BLE v4.2 [Blu14] chip, we investigate the behavior of device discovery and data transmission, in order to increase the performance of the system. Both device discovery and data transmission are dependent on several parameters, that are set at compile time and, usually, not changed afterwards. Depending on the selected operating system (e.g., Android or iOS), devices may support only few different settings, limiting the possibility of optimally fitting the requirements. Furthermore, developers are not aware of the different setting and their possible combinations, leading to systems that are using default values and delivering suboptimal performance. As a consequence of using static default parameters, most BLE devices are either consuming more energy than actually needed or being very unresponsive because the settings are too energy conservative.

These problems are strengthened by the fact that application requirements can change over time and can be user dependent. By using static parameter settings, previously made assumptions about application requirements may no longer be met. For example, during the day a system needs to be highly reactive as users want to connect frequently to the device, but during night the system is not used at all. This assumption can change if we investigate the user behavior. Some users may use a device during night, others during the day. Developers of the specific application have to be aware of such a shift of requirements in order to save as much energy as possible without diminishing the user experience. This time and user dependency of application requirements introduce the necessity of a dynamic adaption of protocol parameters.

1.2 Contribution

We model the BLE device discovery process (advertising and scanning) in detail in order to gain a deep understanding of its influence on energy consumption and responsiveness of the system. Furthermore, the model is used to estimate the expected energy consumption and latency of the overall system. In contrast to the common opinion, device discovery does not depend only on parameters used while advertising, but also on scanning parameters. Therefore, we introduce Optimal Scan Parameters (OSP), a mathematical approach that determines the most suited scan parameters based on the advertising parameters of the remote device such that the device discovery latency is as short as possible.

Based on the BLE device discovery model, we propose Adaptive Advertising (AA), a technique which allows a BLE device to learn from its past activities and adapt the radio's physical parameters accordingly. The algorithm is generic and can be used for different application scenarios. Among others, it offers the following features:

- variable number of advertising levels adjustable at compile time;
- variable calculation granularity adjustable at compile time;
- variable dividers in order to reduce time spent on each advertising level, minimizing the consumed energy, adjustable at compile time;
- variable number of user data adjustable at compile time;
- implementation in C without the need of any external library, making it platform independent, and
- simple algorithm executed during runtime on a daily basis directly on the micro controller, i.e., no data has to leave the system.

Adaptive Advertising allows to increase the performance of a BLE system in periods of high activity, while saving energy whenever possible.

To increase the responsiveness of BLE even further, we introduce the concept of Range Extender (RE). The RE is an additional BLE device that informs nearby interested devices about the approach or appearance of other BLE devices. This allows devices to save energy by setting their own parameters accordingly and adjust them whenever they get notified by the RE about possible necessary interactions.

The RE functionality is implemented as generic solution and is designed in two different modes. The difference between modes is on how the information exchange between RE and interested devices takes place. In the first mode a permanent connection (PC) between two devices is maintained. In the second mode a temporary connection (TC) is established once information has to be exchanged and terminated afterwards. Depending on the application scenario, one mode may be a better option. Both scenarios are evaluated accordingly in order to understand which mode is better suited for a specific scenario.

The RE offers the following features:

- two different modes allowing to react on device constraints and application needs;
- variable connection interval, adjustable at compile time in order to meet an application's power and latency constraints;
- variable number of supported connections;
- variable scan settings adjustable at compile time;
- event-driven scanning;
- variable advertising settings adjustable at compile time;
- variable duration of storing devices adjustable at compile time;
- implementation in C without the need of any external library, making it platform independent, and
- implementation using Zephyr OS on Nordic Semiconductor's nRF52.

We evaluate AA and RE with respect to power consumption and device discovery latency on a real world application: the Nuki Smart Lock (NSL). The NSL allows to lock/unlock a door with a smart phone or a small key fob. For this application, a reasonable trade-off between energy consumption and user experienced latency has to be found. Both energy consumption and latency depend on the discovery time of a user's device and the NSL. We show that AA and RE help decreasing this time and reducing the device discovery latency of the system.

AA is realized on the CY8C4248LQI-BL483 chip, the same as the NSL is using. In contrast, RE is build on top of the operation system Zephyr on Nordic Semiconductor's nRF52. Additional hardware for the power consumption measurement of both chips is used.

Compared to static advertising, our measurements show that AA is able to reduce the user experienced latency by more than 50 % while consuming half of the energy. The RE reduces the device discovery time up to 95 %.

1.3 Outline

The remainder of this thesis is structured as follows. Chapter 2 introduces BLE v4.2 and describes the important parts of the stack that are needed to understand the contributions of this thesis. Furthermore, the Nuki Smart Lock and other used hardware are introduced, highlighting the particularities and describing the setup of each device. Moreover, we discuss the real time operating system Zephyr and its advantages. Chapter 3 lists some existing smart locks that are using similar approaches as the Nuki Smart Lock. Furthermore, we mention already existing work in the field of energy consumption, device discovery, and runtime parameter adaption of BLE. Chapter 4 models BLE device discovery, starting with advertisements and its energy consumption as a function of the amount of advertising payload. Therefore, a simple mathematical model is introduced. Furthermore, the line of sight range is measured and the maximum time for device discovery is estimated. Finally, we introduce Optimal Scan Parameters, a mathematical approach to determine the optimal scan parameters for a given advertising interval. In Chapter 5, we introduce Adaptive Advertisement, a smart advertising technique that helps to decrease the device discovery latency and the energy consumption. It does so by learning from user behavior and adapting the advertising interval of a BLE advertiser at runtime. We discuss and evaluate the algorithm in detail according to the achieved mean device discovery latency and the mean power consumption. Chapter 6 explains the concept of the Range Extender that notifies interested BLE devices about the presence of other BLE devices. The description of the main concept is followed by an elaborated discussion and evaluation, showing the mean device discovery latency and the mean power consumption. Chapter 7 concludes this thesis, followed by an overview of possible improvements in Chapter 8.

Chapter 2

Background

This chapter is divided into three parts. Section 2.1 describes background knowledge about Bluetooth Low Energy, its device discovery, connection establishment, data transmission, and security mechanisms. Section 2.2 investigates the used hardware and lists its relevant features. Section 2.3 introduces the operating system Zephyr Project that is used for the RE implementation on Nordic Semiconductor's nRF52 chip.

2.1 Bluetooth Low Energy (BLE)

Bluetooth was developed before the turn of the millennium by the Bluetooth Special Interest Group (SIG) [BS17] that includes nowadays more than 30000 companies. The idea was to define a global standard for short-range, low-power, and low-cost cable replacement enabling communication between devices [Gup13]. Today, Bluetooth can be found in almost every mobile phone, tablet or laptop. With the accelerated growth of the Internet of Things (IoT) [GBMP13] and its requirements, such as energy efficiency and security, Bluetooth Low Energy (BLE) was developed. With the introduction of BLE, also called Bluetooth Smart, a new chapter was added to the Bluetooth story. Since the end of 2009, BLE was added as a part of the Bluetooth 4.0 specification. Since then, three new specifications have been published: 4.1, 4.2 and 5.0. In this section we mainly focus on specification 4.2. For simplicity, BLE according to Bluetooth Specification 4.2 is tagged as BLE v4.2. This applies also for other specifications.

The success of BLE can be attributed to its capability of targeting the ultra low-power constraints of the new devices, which enables a device to operate for months or even for years on one coin cell battery. This allows to market BLE devices in key application domains, such as:

- health care,
- sports and fitness,
- smart cities, and
- home automation

In order to achieve low power operations, BLE is a mostly-off technology [Spo16]: the radio is disabled most of the time, resulting in a low energy consumption. Once needed,

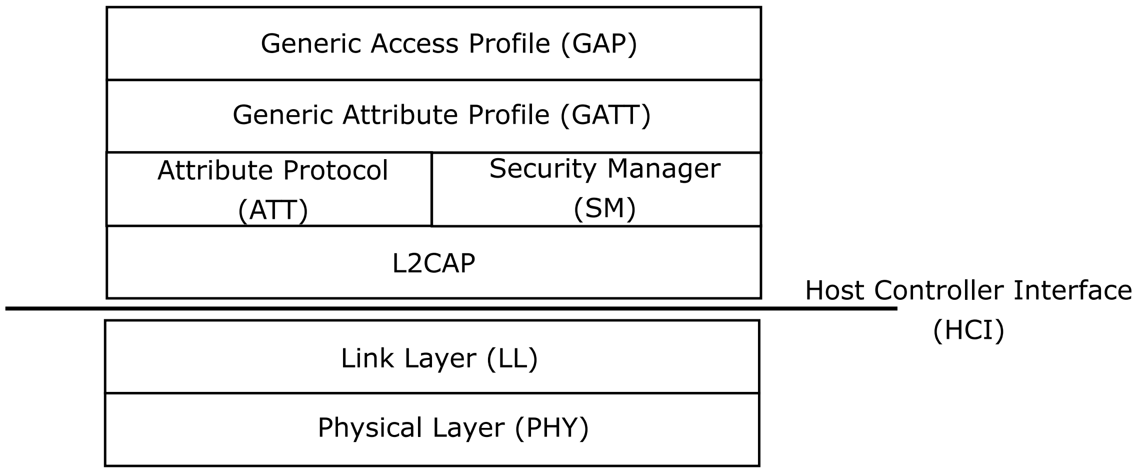

Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Attribute Protocol (ATT)
Security Manager (SM)
L2CAP
Host Controller Interface (HCI)
Link Layer (LL)
Physical Layer (PHY)

Figure 2.1: BLE stack and its different layers (adapted from [Gup13]).

the radio is enabled and communication can take place. In BLE, it can be distinguished between two types of data communication: connection-based and connection-less.

Connection-based means that two devices have to establish a connection in order to exchange data (see Section 2.1.2.5). Once they are connected, data can be exchanged bidirectionally on the data channels of BLE. Transmitted data can be encrypted and the receiver can be addressed directly. Theoretically, the size of the transmitted data is not limited, as large data is splitted into smaller, same sized packets. Those packets are sent and reassembled on the receiver side (see Section 2.1.4). The limiting factor is the memory of both devices. Connection-based data transmission is reliable. This means that every transmitted packet gets acknowledged by the link layer. If a packet is lost or corrupted, the packet gets re-transmitted automatically. The number of re-transmission is not limited: the attempt of re-transmitting stops on link loss. Therefore, this mode is used when bidirectional communication is necessary and a large amount of data or sensitive data has to be transmitted.

BLE allows to exchange data also without establishing a connection, with its connection-less mode. Two devices that are not connected can exchange unidirectional data by using the three advertising channels (see Section 2.1.2.1). The sending device needs to advertise on one to all three channels, the receiver has to scan on the used channels (see Section 2.1.2.2). Data is structured into so called advertising packets (see Section 2.1.2.3). The number of transmitted bytes of payload in one advertising packet is at most 31 (BLE v4.2 and earlier). Note that advertising packets can be received and read by all scanning devices, as there is no encryption for advertising packets. Therefore, this mode is used for broadcasting small amounts of nonsensitive data.

The interoperability between devices of different vendors is guaranteed by implementing the Generic Access Profile (GAP) and the Generic Attribute Profile (GATT). Therefore, their implementation is mandatory for all BLE devices. The profiles define the basic concepts of BLE regarding device discovery and data transmission. GAP defines procedures for device discovery, connection establishment, and security features. In contrast to GAP, GATT defines how devices exchange data.

In the next sections the focus is on the BLE stack, divided into different layers and illustrated in Figure 2.1. We discuss all layers, starting from the bottom and focus on the relevant parts in order to understand the contribution of this thesis.

2.1.1 Physical Layer (PHY)

The Physical Layer (PHY) is the bottom layer in the BLE communication stack (see Figure 2.1). Its responsibility is sending and receiving data over the air.

BLE operates in the 2.4 GHz industrial, scientific and mechanical (ISM) band. Data modulation is performed using Gaussian Frequency Shift Keying (GFSK). As the priorities are shifted towards energy efficiency, the data rate of the physical layer is limited to 1 Mbps (BLE v4.2). In contrast to classic Bluetooth (BR/EDR), BLE uses only 40 channels, each 2 MHz wide, where three channels (37, 38, 39) are reserved for device discovery, also called advertising channels. In order to allow a successful device discovery, interference is kept to a minimum by selecting the three channels in between of the three most popular Wireless LAN channels (1, 6, 11). Limiting the number of advertising channels enables a faster connection establishment, allowing to react faster and saving energy, as the radio has to be turned on shorter. The remaining 37 channels are data channels that are only used once a connection between two devices is established.

2.1.2 Link Layer (LL)

In the BLE communication stack hierarchy (see Figure 2.1) the Link Layer (LL) is on top of the Physical Layer. It is responsible for establishing and controlling links, and exchanging data. Furthermore, the LL manages the frequency selection: BLE uses Adaptive Frequency Hopping (AFH) on the data channels, which allows to reduce interference from other devices in the ISM band.

In order for two BLE devices to exchange data, at least one device has to be aware of the other device. This procedure is called device discovery. In BLE, device discovery is handled using advertising and scanning events. Both advertising and scanning are described next.

2.1.2.1 Advertising

The device advertising its presence is called advertiser. It is the device that wants to be discovered and thus it is transmitting its presence on the advertising channels. Advertising is performed periodically in so called advertising events. The advertising procedure is illustrated in the upper part of Figure 2.2.

During an advertising event an advertiser is sending advertising packets that takes t_{ADV} time, followed by a time of radio inactivity till the next advertising event starts. During this time t_{DS} the radio can be switched off and the system may be put in deep sleep mode. The time between the start of two consecutive advertising events is called advertising interval (T_{ADVI}). At the end of each advertising interval, a random delay between 0 and 10ms (t_{Delay}) is added. This delay should avoid persisting collision with other BLE devices.

The advertising interval can be set between 20ms and 10.24s. As in BLE three out of 40 possible channels are used as advertising channels, there is the possibility to advertise

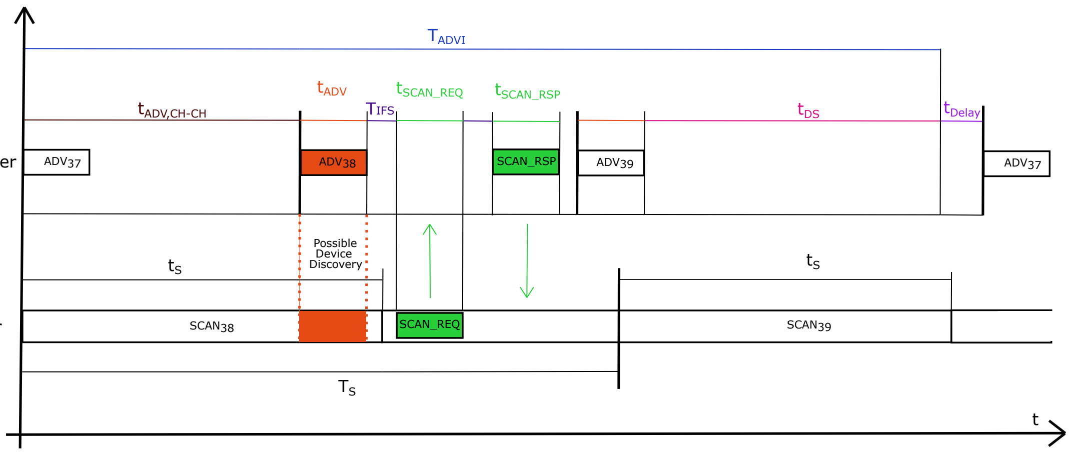


Figure 2.2: BLE device discovery: advertising (upper part) and active scanning (lower part) on three channels with successful device discovery on channel 38.

on any combination of all three channels (channel 37, 38 and 39) during a single advertising event. When a device supports requests (scan and connection requests, described in Section 2.1.2.2 and 2.1.2.5) each transmission phase of an advertising event on one channel has to be followed by listening on the corresponding channel for a possible response of another BLE device. Afterwards, the radio switches to the next channel (inter-channel transition). The time spent on one channel (transmitting, listening and switching to the next channel) is considered to be $t_{ADV,CH\text{-}CH}$.

2.1.2.2 Scanning

On the one hand, using multiple advertising channels increase the robustness against radio interference of other communication technologies. On the other hand, a device that wants to detect the advertiser needs to scan on all advertising channels. As the radio can only scan a single channel at one point in time, the device has to perform alternating scanning. The scanning procedure is illustrated in the lower part of Figure 2.2. We distinguish between the scan interval (T_S) and the scan window (t_S). The scan interval determines the time between the start of two consecutive scanning phases. The scan window determines the net scanning time where the radio is in receiving mode. If T_S is equal to t_S, we talk about *continuous scanning*, as the radio is never turned off. Still, switching between channels has to be performed. This switching takes several tenths of microseconds depending on the used BLE chip. Roughly said, an advertising packet will be received successfully if t_{ADV} and t_S of the same channel overlap. This is illustrated with the dotted lines in Figure 2.2.

The scanner is aware of the presence of the advertiser once an advertising packet is received. Depending on its type (see Section 2.1.2.3), the scanner has three options: sending a scan request (SCAN_REQ), sending a connection request (CONNECT_REQ), or continue listening.

By sending a scan request, the scanner can ask for further advertising information. Sending a scan request is also known as *active scanning*. If a scan request is received, the advertiser may answer with a scan response. Afterwards, both devices are aware of each

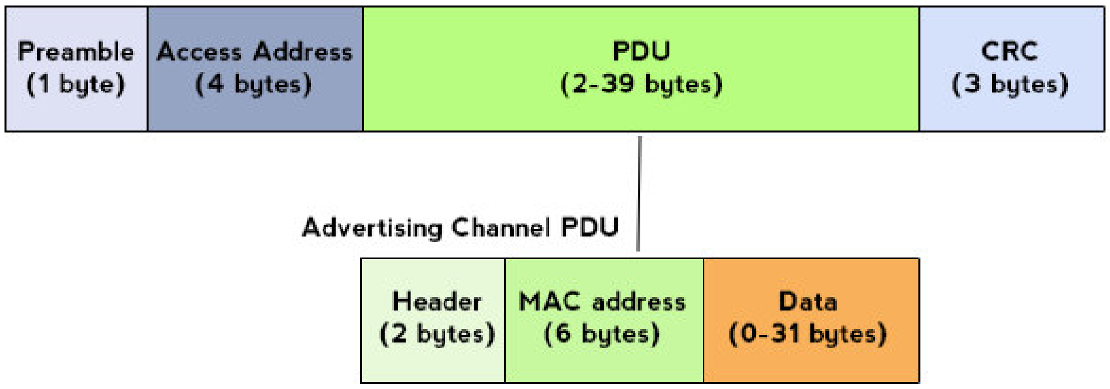


Figure 2.3: Structure of BLE advertising packet according to BLE v4.2.
Source: `https://os.mbed.com`

other. The scenario is illustrated in Figure 2.2. The usage of a scan response is useful in order to ask for further information in case not all data could fit in the advertising packet. Note that, with this method, the size of the transmitted data can only be doubled. If more data has to be transmitted, a connection has to be established.

If the scanner wishes to establish a connection, it sends a CONNECT_REQ. More information about connection establishment can be found in Section 2.1.2.5.

The last option for the scanner is to simply continue listening, called *passive scanning*. This allows the scanner to observe passively the environment not enclosing information about its own presence. Still the scanner is able to establish a connection, whenever needed. Please note that, using passive scanning does not inform the advertising device about the presence of another BLE device.

Regardless of the option chosen by the scanner, all requests have to be performed on the channel where the advertising packet was received, respecting the Inter Frame Space time (T_{IFS}) of 150 μs. T_{IFS} defines the interval between two consecutive packets on the same channel. Therefore, the advertising device has to switch to listening mode after every advertising packet.

2.1.2.3 Advertising Packet Structure

In each advertising event an advertising packet is transmitted up to three times (once on every channel). This packet has a predefined structure, that is illustrated in Figure 2.3. We can observe that even without sending any data in the advertising packet we have to send 10 bytes overhead. Depending on the advertising type, the actual payload structure (data) changes. The advertising types differ in connectivity (advertiser accepts connection requests), directness (advertiser wants to address only a specific device by including its address) and if the advertiser allows scan requests. This results in four advertising types:

1. ADV_IND (connectable undirected): the device accepts connections from any device.
2. ADV_DIRECT_IND (connectable directed): the device accepts connections from a specific device only.

Android Version	Advertising [ms]	Scanning [ms]	
	T_{ADVI}	T_s	T_S
4.3 & 4.4	not possible	200	200
5.0+	100	500	5000
	250	2000	5000
	1000	5000	5000

Table 2.1: Android's possible settings for all versions with BLE support: advertising interval (T_{ADVI}), scan window (T_s), and scan interval (T_S).

3. ADV_NONCONN_IND (non-connectable undirected): the device does not accept any connections.
4. ADV_SCAN_IND (scannable undirected): the device accepts scan requests and connection requests from any device.

For all cases, the advertiser's address is included into the payload, which reduces the actual payload by 6 bytes. This sums up to 16 bytes of mandatory overhead. Therefore, the actual payload of an advertising event is 31 bytes. As already mentioned in Section 2.1.2.2, this can be doubled by using the scan request/response principle. When using directed advertising (ADV_DIRECT_IND), we have to include the scanner's address, which reduces the amount of payload by another 6 bytes.

2.1.2.4 Parameters on Smart Devices

BLE is available on everyday devices such as smart phones and tablets. Most of these devices run Android or iOS: both limit the possibilities of different advertising intervals, scan intervals and scan windows.

Android. Android introduced BLE with version 4.3 (2013). With version 5.0, a radical revision of the Bluetooth implementation took place. Therefore, we have two different settings, illustrated in Table 2.1.

Before the introduction of Android 5.0, no advertising was possible. Devices could only act as connection master. Updating the device to a newer version does not change this ability. With the release of Android 5.0 three different advertising intervals are supported [Chi17]:

- ADVERTISING_INTERVAL_LOW_MILLS (100 ms)
- ADVERTISING_INTERVAL_MEDIUM_MILLS (250 ms)
- ADVERTISING_INTERVAL_HIGH_MILLS (1000 ms)

Scanning was available since the introduction of BLE. In version 4.3 and 4.4 there was only continuous scanning with a scan interval of 200 ms possible. With the introduction of Android 5.0, the scan interval is set to 5000 ms and three different scan windows are supported, allowing also non-continuous scanning. The three options are tagged as [Chi17]:

- SCAN_MODE_LOW_POWER (500 ms)

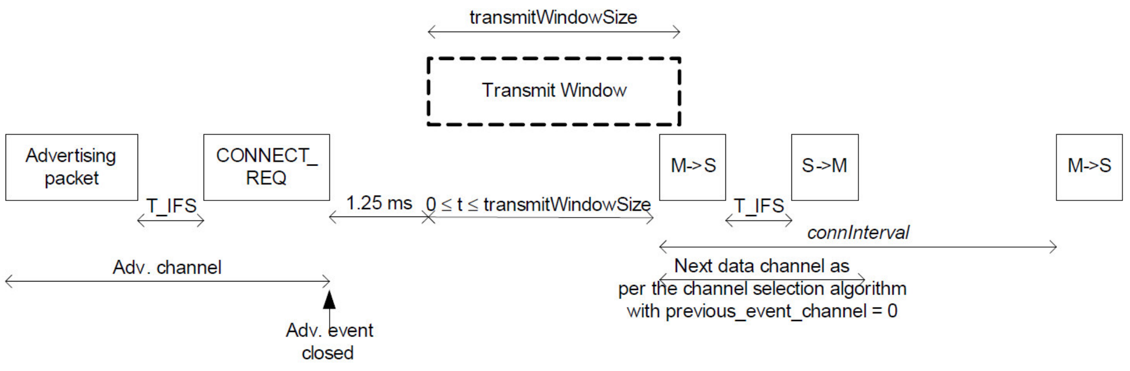


Figure 2.4: BLE connection establishment [Blu14]. The scanner turns into the master of the connection (M), the advertiser acts as slave (S).

- SCAN_MODE_BALANCED (2000 ms)
- SCAN_MODE_LOW_LATENCY (5000 ms)

iOS. Apple's iOS introduced BLE with iOS 5, which was released in 2011.

- Advertising: a lot of different advertising intervals are possible. Apple suggests to advertise with an interval of 20 ms for the first 30 seconds and to switch then to some special slower advertising intervals. More information can be found in Apple's developer guidelines [Inc17a].

  The advertising options depend on the current state of the application: running in the foreground or in the background. For the scope of this thesis this is not relevant.

- Scanning: again, iOS distinguishes between two different scanning policies that depend on whether the app is running in the foreground or in the background. The scan interval and window can not be changed and are selected considering performance and power consumption. Therefore, when the application runs in the background scanning with a lower duty cycle is used in order to conserve energy. This means that the scan window is considerably shorter than the scan interval. Unfortunately, it was not possible to find exact values for both interval and window. More information can be found on [Hid17].

2.1.2.5 Connection Establishment

Once device discovery is successful (see Section 2.1.2.2), and the advertising device allows connection establishment, the scanning device can initiate a connection by sending a connection request (CONNECT_REQ) to the advertising device (see Figure 2.4). The scanner turns into the master of the connection, the advertiser acts as slave. The connection request packet is sent on the advertising channel where the advertising packet was received and contains connection parameters defined by the master. Although the master defines the first connection settings, they may be (re-)negotiated later. Both devices have to agree on the connection parameters and can request a connection parameter update

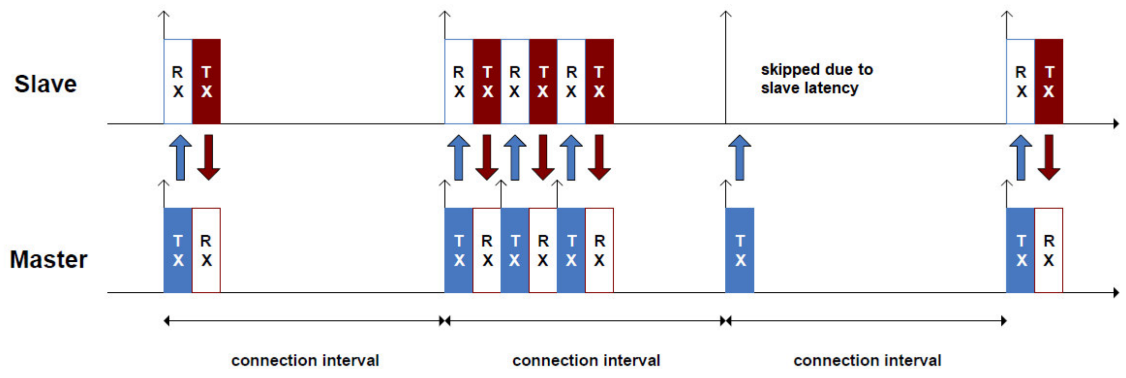


Figure 2.5: Overview of several BLE connection events. The master initiates each connection event. The slave has to respond to each packet. The slave latency allows the slave to skip connection intervals.

Source: Adapted from `https://devzone.nordicsemi.com`

once the connection is established (see Section 2.1.2.6). The connection parameters of the CONNECT_REQ are connection interval (connInterval), slave latency (connSL), channel map, and transmit window size.

The connection interval determines the time between the start of two consecutive connection events and is in the range between 7.5 ms and 4 seconds. Similar to advertising and scanning events, we have connection events that can be seen as point of synchronization between two devices. Each connection event is initiated by the master by sending a packet to the slave. The slave has to respond to each packet received by the master. During a connection event master and slave alternate sending and receiving packets. All communication of one connection event takes place on the same data channel. The standard defines no limit on the amount of data packets per connection events, but, in practice, it is limited by the BLE devices. Limiting factors are memory constraints of both devices and time constraints as a master may have to maintain connection to multiple slaves.

The master has to ensure that a connection event terminates at least T_{IFS} (150 μs) before the start of the next connection event. If both devices do not have more data to send, the packet of the slave determines the end of the connection event.

A slave latency value different than zero allows the slave to skip up to that number of consecutive connection events (see Figure 2.5). This gives the slave the possibility to save energy.

The channel map determines which channels are used and defines a hop increment that is a random value between 5 and 16. After each connection event AFH is applied: the hop increment is added to the current channel number resulting in the next channel number. This increases the robustness: some channels may be blocked by other devices operating in the 2.4 GHz ISM band. Switching between channels increases the probability that successful communication can take place.

After sending the CONNECT_REQ the master has to wait at least 1.25 ms, followed by the so called transmit window size. During this time the scanner has to send its first packet to the advertiser, that, in case CONNECT_REQ was successfully received, listens for it. This communication takes already place on one of the 37 data channels.

2.1.2.6 Connection Parameter Update

Central and peripheral devices agree on connection parameters during the connection establishment (see Section 2.1.2.5). Still, there is the possibility to request an update of those parameters during an ongoing connection in any connection event. This is useful if the application requirements change at runtime. For example, there might be the case that a higher data throughput or a lower latency is necessary. Once application requirements changed, previously made assumptions about suitable connection parameters may not be valid anymore. In this case, slave devices are allowed to request an update of the connection parameters. The device that initiated the procedure sends a request with a proposal of new values that can be accepted or declined by the master of the connection.

2.1.2.7 Connection Termination

In BLE, both devices are allowed to close the connection by sending an appropriate packet. Furthermore, a connection can terminate in a unforeseeable way: a device can move out of range, run out of power, or interference is too strong making communication impossible. In those cases, the communication between both devices is interrupted. To be able to detect link loss, both devices are using timers. During the connection establishment, the initiator, after sending the connection request, waits $6\cdot$ connInterval. If after that period no packet was received successfully, the connection is considered lost. Furthermore, BLE defines a connection supervision timeout (connSupervisionTimeout), used by both devices in order to detect link loss after a connection was successfully established. This parameter sets the maximum time between two received packets before the connection is considered to be lost. On each successful reception of a packet the timer is reset. This parameter can be set device specific.

Once the connection is lost, both device may try to reestablish the connection by going back to advertising and scanning state respectively.

2.1.2.8 Address Types

In the world of BLE, a device is identified by its unique 48 bit device address (BD_ADDR), also called public device address. The most-significant 24 bits are the Organizationally Unique Identifier (OUI). The OUI is assigned by Institute of Electrical and Electronics Engineers (IEEE) and is used to determine the manufacturer of a device. The remaining 24 bits are assigned by the vendor of the device. BD_ADDR can not be changed and therefore, once known, it could be used for tracking a device and its user. This is a problem regarding privacy issues. Therefore, BLE introduced random device addresses that can be used alternatively in order to hide the identity of a device while still being able to communicate. A random address changes periodically (developer selectable) and, therefore, the device can not be tracked. Using those private addresses, a BLE device can not determine if another device was present or connected previously, which can be a problem for certain applications. Therefore, a third type of address was introduced: the resolvable private address. Similar to random addresses, this address changes periodically and hides the real identity of a device. In contrast to the random address, devices that are in a relation of trust are able to resolve the BD_ADDR of the peer device. This process is called address resolution and is explained in Section 2.1.5.2.

2.1.3 Host Controller Interface (HCI)

The host controller interface provides a standardized communication mechanism between the lower and the upper part of the BLE stack (see Figure 2.1). Most of the time, the lower layers are located on a separated BLE chip while the upper layers are part of the host [Gup13]. This layer is not mandatory.

2.1.4 Logical Link Control and Adaptation Protocol (L2CAP)

The responsibility of the L2CAP layer is to manage the logical link that is shared between higher layer protocols [Spo16]. It ensures correct packet segmentation and reassembly for the upper layers in the BLE communication stack (see Figure 2.1).

Furthermore, the L2CAP layer defines the maximum transmission unit (MTU) between two connected devices. The actual value is variable and has to be negotiated between both devices during the connection establishment (see Section 2.1.2.5). In BLE v4.2, the minimum MTU size is 23 bytes: an upper boundary does not exist. A smaller MTU size keeps the packet size small: this allows to minimize the sending and receiving buffers [Gup13].

2.1.5 Security Manager (SM)

The Security Manager (SM) layer is located above the L2CAP layer (see Figure 2.1) and provides procedures for pairing, bonding, authentication, and encryption between BLE devices [Gup13]. Those features are needed once security for a specific connection is needed (e.g., transmission of sensitive data). The SM dispenses a key management system that allows to generate and store various keys that may be exchanged with other devices once needed. Furthermore, the SM takes care of the address generation and resolution. The different types of BLE device addresses are discussed in Section 2.1.2.8.

2.1.5.1 Pairing and Bonding

Once two devices want to exchange sensitive data, they have to create a trustful connection between each other. This process is called pairing. During the pairing process both devices exchange the necessary information in order to establish an encrypted connection. A pairing process can be initiated independent by the device role (master/slave) after a successful connection was created (see Section 2.1.2.5). After both devices disconnected, the pairing information is lost and the pairing process, if needed, has to be started again after the next connection establishment. This can be avoided by using bonding procedure. During bonding the exchanged pairing information is stored on the devices so that the information does not need to be exchanged again once the connection is reestablished.

2.1.5.1.1 Security Issues As mentioned, pairing enables the possibility of establishing a secure link. As always, there are some weak points that an attacker can make use of. Although this thesis does not focus on BLE's security features, it is still worth mentioning, as it is of high relevance for the NSL use case.

We can distinguish between three main security issues when talking about BLE communication [Gup13]:

1. Passive eavesdropping: a third device (Eve) listens to the ongoing communication between two devices (Alice, Bob). Eve does not take part of the communication process, therefore Alice and Bob can not notice the presence of Eve. In order to counter this, BLE encrypts the communication over a secure link by using the block cipher AES in Counter Mode with CBC-MAC (AES-CCM), enabling confidentiality and integrity. As every block cipher, AES uses symmetric keys. Only with the knowledge of the key, the message can be encrypted. Therefore, the key has to be exchanged between Alice and Bob.

   This key exchange procedure, in BLE called pairing procedure, can introduce vulnerabilities leading to the scenario in which Eve is able to decrypt the communication between Alice and Bob.

2. Man in the middle (MITM) attacks: a third device (Eve) impersonates the other two devices (Alice, Bob) in order to trick them. Both Alice and Bob establish a connection to Eve, thinking they are connected to each other. By holding both communication keys, Eve can read and also forward the data resulting in the scenario where Alice and Bob do not notice the presence of Eve.

3. Device/identity tracking: ability to physically track a device by its device address. As countermeasure, BLE introduced different address methods (see Section 2.1.2.8). During the pairing/bonding procedure the information for address resolution can be exchanged, enabling the possibility to track a device.

2.1.5.1.2 Pairing Procedure Once a device requested a pairing procedure, both devices exchange a Temporary Key (TK) that is used to create a Short Term Key (STK). This procedure is called *LE legacy pairing.* The connection is encrypted using the STK. The security of this process depends on the selected pairing method, that determines how the TK is exchanged.

After the pairing request, both devices exchange device capabilities and requirements (unencrypted!):

- I/O capabilities: information about the presence of I/O devices like displays, buttons, or keyboards.
- Out of Band Information (OOB): information about the support of any other wireless communication standard that could be used in order to exchange the pairing information (e.g., NFC).
- Maximum key size: each device informs its counterpart about the maximum supported key size.
- Authentication requirements: connection requirements such as encryption, authentication, and signature.
- Bonding requirement: flag stating if the pairing information should be stored or not.

Once both devices are aware of the capabilities of its counterpart, they determine how they are going to set up a secure connection. Now, both devices generate and exchange TK, depending on the selected method:

- Just Works,
- OOB Pairing,
- Passkey, and
- Numeric comparison.

Once a pairing method is selected, depending on the devices' capability, the devices exchange a key in order to encrypt the connection. This indicates the end of the pairing procedure and the start of the optional bonding procedure.

Depending on the BLE version, some methods are more or less secure. More information about the exact exchange of keys and security against passive eavesdropping and MITM can be found in [Bon16] and in the corresponding BLE standards.

With the introduction of BLE v4.2 [Blu14], the concept of *LE Secure Connections* was introduced. This enables the exchange of a Long Term Key (LTK) that is used instead of TK and STK, in order to encrypt the connection. This key is created and exchanged using Elliptic Curve Diffie Hellman (ECDH). Both devices generate a key pair, consisting of public and private key. The usage of this asymmetric keys makes it possible to authenticate the connection: the devices exchange the public keys and compute the shared secret, used as symmetric key in order to encrypt the communication needed for the selected pairing method. After a successful pairing method, the LTK is generated, exchanged and used for further communication. This approach offers stronger security compared to the original BLE key exchange protocol. Therefore, most BLE applications that were developed before the release of BLE v4.2 adapted their own key exchange mechanism.

2.1.5.1.3 Bonding Procedure After the successful pairing of two devices, a bonding procedure can start if it was set as option during the pairing request (see Section 2.1.5.1.2). In this phase, devices exchange several parameters allowing to encrypt the link and in order to avoid another pairing procedure at the next connection establishment. Therefore, a 128 bit Long Term Key LTK is distributed. This key is used to encrypt all further messages. Furthermore, the so called Identity Resolution Keys IRK (128 bit) and the public BLE addresses (BD_ADDR) of the devices are exchanged: they allow to perform address resolution (see Section 2.1.5.2).

2.1.5.2 Address Resolution

The procedure to derive the BD_ADDR of a device from one of its used resolvable private addresses is called address resolution. The different address types of BLE are explained in Section 2.1.2.8. The resolvable private addresses are generated and resolved by the usage of the Identity Resolution Key (IRK). This 128 bit key can be assigned to the device or randomly generated and is held in the key storage of the SM.

In order to map a resolvable private address to a specific device, the resolving device needs to know the BD_ADDR and k_{IR} of the specific device. Therefore, the privacy concept only protects against devices that do not possess the device specific IRK and the corresponding BD_ADDR. In order to exchange those values and create a circle of trust, the bonding procedure has to be performed (see Section 2.1.5.1.3).

This step should be taken carefully: in a scenario where device A and B perform the bonding procedure, device A can create private resolvable addresses based on the received information and impersonate device B. Furthermore, device A could forward the received information to a third device C. This would include C into the circle of trust between A and B, even if B did not agree on that.

2.1.6 Attribute Protocol (ATT)

The Attribute Protocol (ATT) is above the L2CAP layer in the BLE communication stack (see Figure 2.1). It provides procedures for discovering, reading, and writing attributes of a connected device. In BLE, an attribute is an abstract type of data representation and can represent a service, a characteristic, or a descriptor. How attributes are related to each other is defined in the higher layer Generic Attribute Protocol (GATT), described in Section 2.1.7.

2.1.6.1 Attributes

An attribute can be accessed in a client-server model: the server provides a set of attributes to the client. Which attributes a server supports and their exact structure have to be discovered by the client. Depending on different permissions, the client may be allowed to read and/or write attributes. Furthermore, the server can notify or indicate clients about changes of attributes.

At any point in time, only one server can be active on a device. Still, the device can implement a client role, a server role, or both client and server role [Gup13].

An attribute can be seen as something that represents data, so a value or information about the value. It is a discrete value containing:

- attribute type: each attribute is uniquely defined by a UUID. In general, BLE is operating with 128-bit UUIDs, but a range of UUID values has been pre-allocated in order to allow devices faster attribute discovery. Internally, every device uses 128-bit UUIDs, that can be derived easily from the shorter UUIDs (16 and 32-bit). The shorter UUIDs are allowed only for services and characteristics defined by the Bluetooth SIG. For custom services and characteristics, the full 128-bit UUIDs have to be used.

- attribute handle: a 16-bit value used to uniquely identify an attribute on the server. It is assigned by each server to its own attributes. Using the handle, the client can perform read or write requests. Furthermore, the handle is used to identify notifications or indications.

- attribute value: value of variable length and data type.

- set of permissions: specifies that an attribute may be read and/or written. Furthermore, necessary security features in order to access a specific attribute can be defined (e.g., encryption).

2.1.6.2 Attribute Discovery

Every client device has to perform attribute discovery to find out the server defined attribute handles. Only by knowing those handles a client can perform requests in order to read/write data. As GATT defines an attribute hierarchy, the client has to start to discover the services first, followed by characteristics and optional descriptors. Once the client is in possession of the attribute handle of the desired characteristic, it can start to perform actions in case the client holds the necessary permissions:

- read: reading a specific characteristic value.
- write: change the value of a specific characteristic value.

2.1.6.3 Notifications/Indications

The notifications/indications feature was introduced in order to avoid data polling which would dramatically increase the energy consumption of client devices. Imagine the scenario where a device C (client) is dependent on the sensor value of device S (server). In order to notice changes of that value, C would need to periodically request the sensor value. This is called polling. Dependent on the polling frequency, the energy consumption of both devices changes, independent from how often the sensor value actually changes. Notifications/Indications avoid that scenario. If S supports notifications or indications, C can subscribe to this service. After a successful attribute discovery, C knows the attribute handles used by S. Therefore, C can enable notifications by sending a write request and setting the corresponding characteristic to 1. Once notifications/indications are enabled, S informs C about changes of the sensor value. The communication is reduced to the minimum, conserving as much energy as possible on both devices.

Of course, a device can disable notifications/indications whenever needed. Therefore, again, a simple write request, now with value 0, is sufficient.

The difference between notifications and indications can be found on application level. In contrast to notifications, indications need an acknowledgment on the application level. This limits the amount of indications to one indication per connection event. The application acknowledgment is sent on the next connection event. Therefore, two consecutive indications are at least two connection events apart. There is no limitation on the amount of notifications per connection interval (see Section 2.1.2.5). Hence, notifications are preferred if a higher transfer rate is needed.

For transmission of both notifications and indications, Attribute Protocol PDUs are used. Their size depends on the MTU both devices agreed on (see Section 2.1.4). The PDU format introduces three bytes of overhead that reduce the actual payload [Blu14]:

- Opcode: 1 byte in order to select between indication or notification. Of course, there exist more opcodes for other ATT features.
- Attribute Handle: 2 byte in order to specify the attribute. The assigned handle is server specific.

This results in an actual payload of **MTU - 3** bytes.

2.1.7 Generic Attribute Profile (GATT)

The Generic Attribute Profile introduces a hierarchy allowing to organize information into profiles, services, characteristics, and descriptors. Services, characteristics, and descriptors are also called attributes (see Section 2.1.6). A profile is composed of one or more services necessary to fulfill a use case. A service is a collection of data associated to a particular function and consists of at least one characteristic or references to other services. As shown in Figure 2.1 this layer is on top of the ATT layer.

Profiles and services are defined by the Bluetooth SIG, but they can also be developer defined. In the second case we talk about custom profiles and services. A list of predefined profiles and services can be found on the official website [BS17].

2.1.8 Generic Access Profile (GAP)

The Generic Access Profile (GAP) defines the base functionality and roles common to all BLE devices. Thus, its implementation is mandatory. GAP defines procedures for devices discovery, establishing and terminating connections, and exchange of security features and data [Gup13]. The profile should ensure interoperability between devices from different vendors.

GAP defines four different roles for BLE devices:

1. Broadcaster: a device is only transmitting advertising packets. It never receives any data, thus only a transmission unit is needed. This role is used for broadcasting data.

2. Observer: a device is only scanning for advertising packets. Thus, it never transmits any data. Therefore, only a receiving unit is needed. This role is used for collecting broadcasted data or to track other devices.

3. Peripheral: a device that accepts incoming connection request, further serving as slave of the connection. The device needs transmission and receiving unit.

4. Central: a device that initiates connections by sending connection requests. The device needs transmission and receiving unit.

Once supported by the used BLE stack, a device can be in all 4 roles at the same time, holding multiple connections.

2.1.9 Bluetooth Low Energy 5.0

In December 2016 the latest BLE version, tagged as 5.0, was released by the Bluetooth SIG [Blu16]. Unfortunately, till now, not many BLE chips are supporting the new version. Even though this thesis focuses on the BLE v4.2 standard, we want to mention the new possibilities that BLE v5.0 offers. The major improvements and new features introduced by BLE v5.0 can be summarized by the following points:

1. Physical Layer **LE 2M**

   BLE v5.0 brings a new physical layer that increases the data rate. Until BLE v4.2 the data rate was limited to 1 Mbps (LE 1M). BLE v5.0 devices should support both layers. With the new layer, doubling the data rate is possible. LE 2M supports a data rate of 2 Mbps. Doubling the data rate means that the transmission and reception time is halved. LE 2M uses the same transmission power as LE 1M. The increased data rate is achieved by a higher data modulation at the price of a smaller link budget. As we know, most of a device's energy demand is produced by an active radio. Thus, a shorter transmission and reception time reduces the energy consumption significantly. This leads to an increased battery life.

2. Physical Layer **LE Coded**

   An optional new physical layer, called LE Coded is introduced by BLE v5.0. With this option, coding with a spreading factor of two or eight is possible. This option is only available on LE 1M. On the one hand, coding increases the redundancy and makes the connection more robust and reliable. On the other hand, it reduces the effective symbol rate. Once an application can disclaim a high data rate, this allows to increase the range of BLE significantly.

3. Transmission Power

   The maximum transmission power for BLE v4.2 was 10 dBm. BLE v5.0 multiplies this value by ten and makes a transmission power of 20 dBm possible. Together with LE Coded, this should increase the communication range by a factor of four.

4. Extended Advertising

   BLE v5.0 adds a new possibility to broadcast data. So far, in BLE v4.2, broadcasting was only possible by advertising packets using up to three advertising channels. The amount of data was limited to 31 bytes of payload per advertising packet. Listening devices could not know in advance on which channels devices are advertising data packets, which made it necessary to advertise the same data on all three channels.

   With BLE v5.0 the concept of extended advertising was introduced. This concept allows advertising devices to use data channels for broadcasting data. The idea is that the device is announcing on the advertising channels that data is sent on a specific data channel. Interested devices can listen on the corresponding frequency at the given time. The advertising packet on the data channel contains information about the next used channel and its transmission time. Furthermore, the amount of payload is increased from 31 bytes to 255 bytes. Once devices miss a certain packet or new devices want to listen to it, they do not know the next used channel. Thus, synchronization is needed: from time to time the advertiser is advertising again on the three advertising channels, allowing all listening devices to follow the upcoming chain of advertising packets on data channels.

   Extended Advertising allows to broadcast bigger amount of data in a more efficient way.

2.2 Hardware

In this section we describe the hardware used for this thesis. The description includes basic information about the device as well as particular properties that are important for this thesis.

(a) Nuki Smart Lock.

(b) Nuki fob.

(c) Nuki bridge.

(d) Nuki box.

Figure 2.6: Different Nuki devices [Gmb17].

2.2.1 Nuki Smart Lock (NSL)

The Nuki Smart Lock (NSL)[Gmb17] allows to lock/unlock doors with the smart phone. The lock is illustrated in Figure 2.6a. The NSL is mounted on the inside of the front door. Almost any double cylinder lock is supported. Putting the physical key on the inside of the door, the NSL is able to turn the key once an user action is received. BLE v4.0 is used as communication protocol. Nuki offers three other devices:

- Key fob: can be used instead of a smart phone in order to lock/unlock the NSL via BLE. The fob is illustrated in Figure 2.6b.
- Bridge: brings the NSL online and allows remote control of the NSL. Furthermore, this enables smart home integration. The bridge supports two communication protocols: IEEE 802.11 for the communication with the home network and BLE for the communication with the NSL. An illustration of the bridge can be found in Figure 2.6c.
- Box: brings the smart lock functionality to the entry intercom of a building, presented in Figure 2.6d.

To control the NSL with the smart phone the Nuki application has to be installed. The app is available for both Android and iOS.

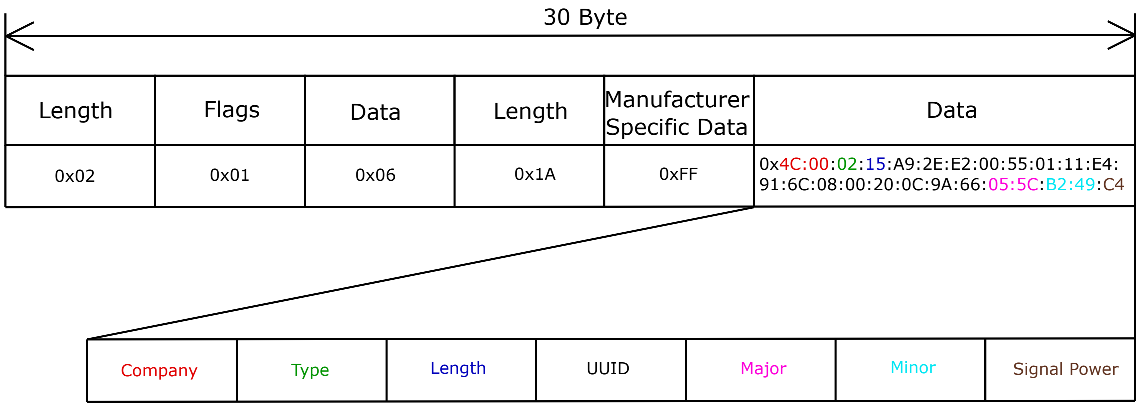

30 Byte
Length
Flags
Data
Length
Manufacturer Specific Data
Data
0x02
0x01
0x06
0x1A
0xFF
0x4C:00:02:15:A9:2E:E2:00:55:01:11:E4:
91:6C:08:00:20:0C:9A:66:05:5C:B2:49:C4
Company
Type
Length
UUID
Major
Minor
Signal Power

Figure 2.7: Advertising packet structure of the NSL. The structure implements the functionality of an iBeacon.

2.2.1.1 General functionality

The NSL acts as advertiser and advertises its presence in predefined time intervals. A user approaches the door and tries to unlock the door with his smart phone or by using the key fob. Normally, the unlocking devices would need to scan for the NSL's advertising packets in order to perform device discovery (see Section 2.1.2). This is only possible once the device of the user is in range of the NSL. Nuki follows a different approach. Once the user wants to perform an action, his device directly sends a connection request using the public device address (BD_ADDR, see Section 2.1.2.8) of the NSL. If both devices are in range, the NSL can react appropriately. If the devices are out of range after a certain timeout, the corresponding message appears on the screen of the smart phone. Therefore, scanning is not necessary. If BD_ADDR of the NSL is not known, a BLE scan would become necessary.

2.2.1.2 Advertising Settings

The NSL implements the functionality of an iBeacon, i.e., it periodically advertises its presence following Apple's iBeacon definition. Therefore, the advertising packet of the NSL must contain:

- UUID: 16 byte Identifier (attribute type, see Section 2.1.6.1) of the service. Nuki advertises its own key turner initialization service UUID [Gmb16].
- Major: 2 byte string in order to identify a subset of beacons offering the same service indicated by the UUID.
- Minor: 2 byte string individually identifying an individual beacon
- Signal Power (optional): 1 byte indicating the expected signal strength exactly in one meter distance of the device. This is a hard coded value and can be used by other devices together with the Received Signal Strength Indicator (RSSI) to roughly estimate the distance.

ADV_{Mode}	$ADV_{Interval}$ [ms]
active	152.5
normal	417.5
slow	1022.5
slowest	2000.0

Table 2.2: Advertising intervals supported by the NUKI Smart Lock.

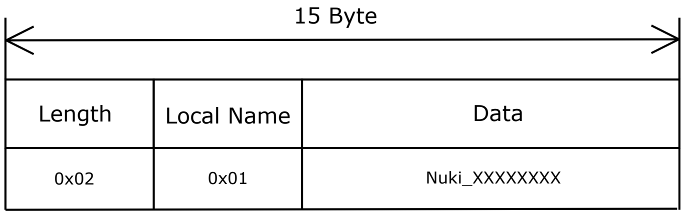


Figure 2.8: Structure of the scan response packet of the NSL, that is sent after every received scan request.

The resulting advertising packet of the NSL looks like the one shown in Figure 2.7. It contains Flags, stating that the device does not support classical Bluetooth and that the device is discoverable continuously without any timeout. Furthermore, the packet contains Manufacturer Specific Data (0xFF) embedding the manufacturer company identifier that is assigned by the Bluetooth SIG. As the iBeacon technology is used, type 0x02 is selected and Apple's manufacturer ID (0x004C) is broadcasted. Keep in mind that the values are little-endian. After the type declaration, the length of the following data is specified (0x15 = 21 bytes). Those 21 bytes contain the fields UUID, major, minor and signal power as described above. In total, the NSL is advertising 30 byte of payload with every advertising packet.

The NSL supports the four different advertising intervals that are listed in Table 2.2.

On each received scan request, a scan response containing the complete local name of the NSL is sent. The local name contains the prefix *Nuki_* and the unique 8 byte Nuki-ID. As shown in Figure 2.8, the total payload used in the scan response is 15 bytes.

2.2.1.3 Security

Although every scanning device is able to receive the advertising packets of the NSL, any communication with the NSL is only possible for trusted devices. This relationship of trust is created during a pairing process with key exchange. This key exchange is not a bonding process as described in Section 2.1.5.1.3. At the time Nuki started with its implementation, BLE v4.2 was not released and thus *LE Secure Connections* not available. Therefore, Nuki implemented their own secure key exchange mechanism. This approach is based on Diffie-Hellmann key exchange, and is similar to the one introduced with BLE v4.2. The Diffie-Hellmann key exchange between NSL and user's device can be initiated by pressing the button on the NSL for 5 seconds. During the pairing process, not only a long term key LTK is exchanged but also an authentication ID (4 byte integer) that is unique for every paired device. The user device stores the ID together with the unique

Nuki-ID, BD_ADDR of the NSL and the LTK. The NSL stores the authentication ID together with the LTK.

Every request of the user device is encrypted with the exchanged LTK. In order for the NSL to be able to decrypt the message, the authentication ID is added (unencrypted!). According to the received authentication ID, the NSL can select the appropriate LTK and decrypt the message.

2.2.1.4 Connection Settings

For encrypted data transmission a connection establishment is necessary. Most of the BLE chips on the market support only one active connection at any point in time. This means that the NSL is not responsive for any other devices while the NSL is holding an active connection. Nuki solved this problem by immediately terminating the connection after each user interaction. After the connection termination, the NSL is advertising with the fastest possible advertising interval (active mode, see Table 2.2) for the next 30 seconds. This reduces the user experienced latency during the next 30 seconds if a new connection establishment would be necessary, meaning another user interaction takes place.

2.2.1.5 Logging

The NSL holds a portion of persistent memory where the last 300 locking and unlocking actions are stored. An entry consists of the action (lock/unlock), the user who was responsible for the action, and a time stamp containing date and time. Once the storage is full, the first entry is overwritten.

According to Nuki, the storage information does not leave the lock, even if the lock is used in combination with the Nuki bridge.

2.2.1.6 Smartphone Application

The Nuki application[1] is available for Android and iOS. It is possible to select between three different advertising modes (normal, slow, slowest) in order to conserve battery power on the NSL. The active mode is used by the NSL after a disconnection and can not be selected manually. All possible advertising intervals are illustrated in Table 2.2.

The application allows to govern multiple smart locks and allows to change the name according to the user's preferences. By swiping left and right, the NSL can be locked/unlocked easily. It is possible to enable a feature called *Smart Actions*, that allows to automatically unlock the door once the user gets in range of the NSL. This approach uses Google location services to assign the NSL a certain location. Once the smart phone of the user gets inside a circular area around that point, the applications starts to send automatic unlock requests. This area is larger than the BLE range. Once in BLE range, the NSL is able to receive the unlock request and can start the unlocking process.

The first paired smart phone becomes the administrator of the lock and is free to add/remove other users, i.e., giving away and revoking lock/unlock privileges. This can be temporary (e.g., several days) or infinite with optional restrictions (e.g., only 1 day per week or only 2 hours per day).

[1]https://play.google.com/store/apps/details?id=io.nuki

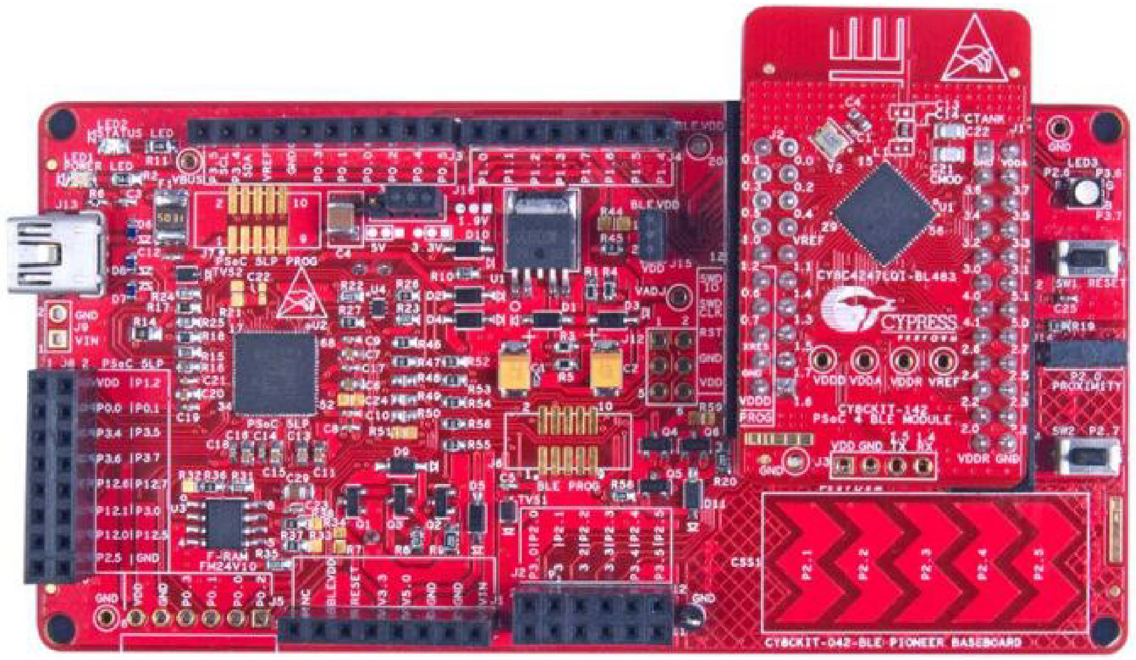


Figure 2.9: CY8CKIT-042-BLE development kit with mounted CY8C4248LQI-BL483 BLE chip from Cypress (CDK).

Source: `www.arrow.com`

2.2.2 Cypress Development Kit (CDK)

During the development, we are using the CY8CKIT-042-BLE development kit from Cypress (CDK). It is a PSoC4 device and part of the PSoC 4200 device family. The CDK allows to connect a variety of shields and expansions boards, that are Arduino or Digilent Pmod compatible or that come directly from Cypress. As BLE chip we used the CY8C4248LQI-BL483 [Cyp17]. The same chip is used by the NSL. The chip runs with a 32-bit ARM Cortex-M0 CPU. It has 256 kB flash memory and 32 kB SRAM. The BLE radio supports BLE v4.2 and its sensitivity goes down to -92 dBm, while supporting an output power starting from +3 dBm down to -18 dBm in 3 dB steps. According to the datasheet the current consumption in receiving mode is 18.7 mA and in transmitting mode 16.5 mA in case a transmission power of 0 dBm is used. Both CY8CKIT-042-BLE development kit and used BLE chip CY8C4248LQI-BL483 are illustrated in Figure 2.9.

2.2.2.1 Setup

Setting up the development environment for the CDK is pretty easy. On the official Cypress website[2], the Integrated Design Environment PSoC Creator can be downloaded. Unfortunately, it is only available for Windows. Once the program is downloaded and installed, the development process can be started. To get familiar with the environment, example programs and videos can be found on the website.

[2]www.cypress.com

The advantage of PSoC creator is the design based approach for the different components of the Development Kit. In a graphical interface, components can be placed and parameter settings selected. Each component has a detailed data sheet and API interface documentation that can be accessed from the IDE. Building the project creates the necessary code. Therefore, only application code has to be written. The code can be debugged with the built-in debugger.

2.2.2.2 Low Power Modes

PSoC 4 devices, like the development kit we are using, support four different low power modes: *sleep*, *deep-sleep*, *hibernate* and *stop* [SK16]. For our application purposes only the first two power modes are relevant:

- Sleep Mode: during this mode the CPU does not run any instruction. All other peripherals remain active and can generate interrupts, that can be used in order to wake up the device.

- Deep-Sleep Mode: in this mode high-frequency clocks and all peripherals that require those clocks are disabled. The low-speed oscillator, the watchdog timer and the watch crystal oscillator remain active. They can be used in order to wake the system up from deep-sleep. Furthermore, the I^{2}C unit can continue in slave mode in order to wake up the device on address match. Additionally, GPIO interrupts can be used as wake up source.

2.2.3 nRF52 Development Kit (nDK)

The nRF52 Development Kit (nDK) is a single board development kit that supports BLE v5.0, ANT, and NFC. Our version is equipped with the nRF52832 SoC. As alternative, the nRF52819 SoC can be used. The used board is illustrated in Figure 2.10. For programming and debugging Segger J-Link OB is used. It is possible to use third party shields compatible to the Arduino Uno Revision 3 standard. This compatibility is useful for the nRF PPK (see Section 2.2.5).

The nRF52832 is running with a Cortex M4F, a 32-bit ARMv7-M CPU. It can be supplied in the range from 1.7 V to 3.6 V. Furthermore, it has 512 kB flash memory and 64 kB RAM. The BLE radio sensitivity goes down to -96 dBm and the chip supports an output power from +4 dBm to -20 dBm in 4 dB steps.

According to the data sheet [Nor17a], the current consumption in receiving mode is 6.5 mA. When used for transmission, the radio consumes 7.1 mA (0 dBm output power). While in ultra-low power mode, a current consumption between 0.3 μA and 1.9 μA, depending on used peripherals, is observable.

2.2.3.1 Setup

We used the nDK in combination with the Zephyr Project (see Section 2.3). Therefore, we describe the setup in combination with the mentioned OS.

In order to be able to flash the nDK over USB, *Segger J-Link* and *nRF5x Command-Line Tools* have to be installed. They can be downloaded from the official webpages `https://www.segger.com` and `http://www.nordicsemi.com` respectively. We use the

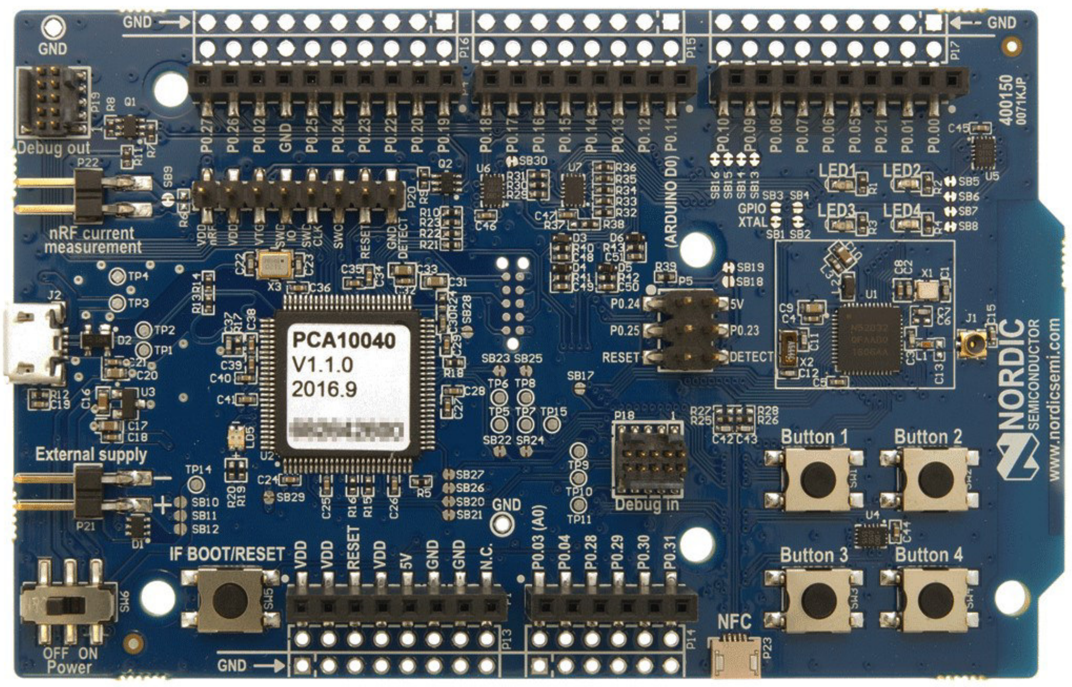


Figure 2.10: Nordic Semiconductor's nRF52 Development Kit (nDK) with nRF52832 SoC.
Source: `http://infocenter.nordicsemi.com`

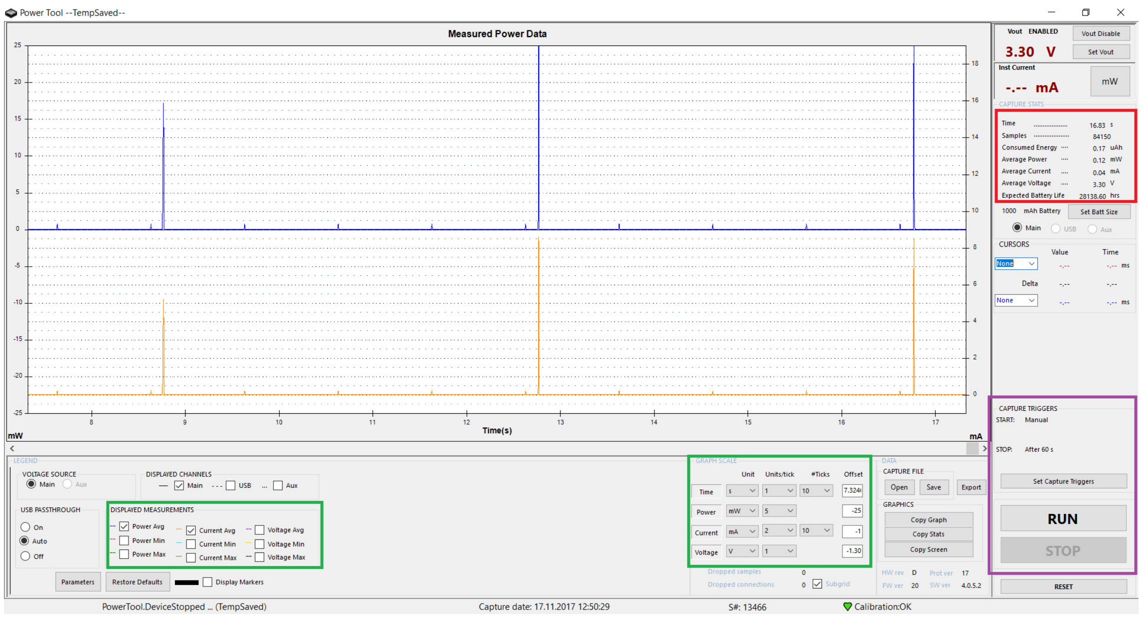
installed command line tools to erase the flash memory and to upload the new program. More details and an accurate description can be found on the Zephyr Project webpage [Fou17] browsing for the corresponding board. If all tools are installed successfully, you can run the following command from your current working directory in order to build and flash a program:

```
make BOARD=nrf52_pca10040 &&
nrfjprog --eraseall -f nrf52 &&
nrfjprog --program outdir/nrf52_pca10040/zephyr.hex -f nrf52 &&
nrfjprog --reset -f nrf52
```

2.2.3.2 Low Power Modes

The nRF52832 supports two different power modes [Nor17a]:

- System OFF: mode where the highest power saving is possible. In this mode, all ongoing tasks are terminated and the core functionality is powered down. A wake up from this mode resets the system. Therefore, this mode is not of use for our use case.
- System ON: in this mode, all functional blocks such as CPU and peripherals can be running or idle, depending on the software sided configuration. Once CPU and all peripherals are in idle mode, the system can enter two sub power modes:



Vout ENABLED
3.30 V

Inst Current
-.-- mA

CAPTURE STATS

Time	16.83	s
Samples	84150	
Consumed Energy	0.17	uAh
Average Power	0.12	mW
Average Current	0.04	mA
Average Voltage	3.30	V
Expected Battery Life	28138.60	hrs

1000 mAh Battery

[✓ Main] USB Aux

CURSORS

	Value	Time	
None	-.--	-.--	ms
Delta	-.--	-.--	
None	-.--	-.--	ms

CAPTURE TRIGGERS
START: Manual
STOP: After 60 s

LEGEND

VOLTAGE SOURCE: [✓ Main] Aux

DISPLAYED CHANNELS: Main ✓, USB ☐, Aux ☐

USB PASSTHROUGH: On [✓ Auto] Off

DISPLAYED MEASUREMENTS

✓ Power Avg	✓ Current Avg	☐ Voltage Avg
☐ Power Min	☐ Current Min	☐ Voltage Min
☐ Power Max	☐ Current Max	☐ Voltage Max

☐ Display Markers

GRAPH SCALE

	Unit	Units/tick	#Ticks	Offset
Time	s	1	10	7.324
Power	mW	5		-25
Current	mA	2	10	-1
Voltage	V	1		-1.30

Dropped samples 0
Dropped connections 0 ✓ Subgrid

DATA

HW rev D Prot ver 17
FW ver 20 SW ver 4.0.5.2

PowerTool.DeviceStopped ... (TempSaved) Capture date: 17.11.2017 12:50:29 S#: 13466 Calibration:OK

Figure 2.11: PowerTool provided by Monsoon Solutions Inc. in order to measure and display the current consumption.

- Constant latency: in this mode, the wake up latency of the CPU and the task response is kept at a constant minimum. Therefore, some resources are kept on, which allows to have a constant and predictable latency at the cost of an increased power consumption.
- Low power: this mode ensures the lowest possible power consumption. The price we have to pay is an increased and unpredictable CPU wake up latency and task response. By default, the system uses automatic power management, which tries to save as much power as possible and, therefore, this mode is used whenever needed.

As we do not have to deal with hard deadlines, we can live with the increased latency and hence, we try to spend as much time as possible in low power mode in order to conserve as much energy as possible.

2.2.4 Monsoon Solutions Inc. Power Monitor

Monsoon Solutions Inc. offers two power monitors: a high voltage power monitor (AAA10F) and a low voltage power monitor (FTA22J). We are using the low voltage power monitor, where a supply voltage between 2.0 V and 4.55 V in 0.01 V steps can be selected [Mon14]. For an accurate current measurement a self-calibrating, integrating system is used. The power monitor has two current ranges which differ in resolution. The proper range is selected during measurement by software. The maximum time resolution that can be used is 20 μs. Once values are written to a file, the average over ten values is build. Thus, the highest logging resolution is 200 μs. The calibration is performed also during runtime, thus self-calibrating. This allows the compensation of temperature changes during measurement run. The current resolution goes down to 2.86 μA with an accuracy of 1 % or 50 μA, whichever is greater.

A .NET software (PowerTool) is available for Windows. Furthermore, a Python library is provided for Windows, Linux, and Mac. Documentation and installation instructions can be found on the official website [Inc17b] and in the manual [Mon14].

In order to perform simple energy measurements, we do not need to use the Python API. The available PowerTool program (see Figure 2.11) allows us to take all measurements we need. Measurements can be taken theoretically infinitely long, the total consumed energy and the average power consumption are displayed on the right side (red box). For the graph, different scaling can be used (right green box). Power, voltage and current can be displayed, if needed, all simultaneously (left green box). Beside a manual start/stop of the measurement, triggers can be used. The trigger selection as well as the start/stop of the measurement can be found inside the purple box on the right side.

The power monitor is used in order to perform all measurements on the CDK (see Section 2.2.2).

2.2.5 nRF Power Profiler Kit (nRF PPK)

Measuring current consumption with Monsoon's Power Monitor is possible for the CDK but not for the nDK. The reason is that the BLE chip can not be supplied in a completely independent fashion from other board components and, therefore, depending on the activities of other components on the board, the results are not sufficiently precise. Nordic Semiconductor provides a Power Profiling Kit (nRF PPK) that can be used to measure current consumption on the development kits nRF51 and nRF52 [Nor17b]. The board is illustrated in Figure 2.12. The nRF PPK board is connected to the top of the development kit, using the Arduino Uno Revision 3 standard pin.

The nRF PPK has an analog measurement unit that allows accurate current consumption measurement in the range from 0.2 μA to 70 mA. Automatic switches ensure a high measurement resolution throughout the whole measurement range. The time resolution goes down to 15 μs, which delivers reliable measurement values as current consumption spikes can be detected.

Nordic Semiconductors provides a PC software for the nRF PPK that comes with a graphical user interface and supports two variable length measurement windows. One for longer measurements (0.1 s to 20.0 s) and one high resolution window (4.82 ms to 26.62 ms) for trigger events. Both windows support averaging functionality. The supply voltage can be selected via software in the range from 1.8 V to 3.6 V. The software is written in Python, which allows customization according to own preferences. Furthermore, there is the option to log longer measurements to a csv file. This allows to go beyond the 20 s measurement window, which would not be enough for our case. We tested the software only on Windows, but it should also work on other platforms.

In order to be able to measure current consumption on the nDK a manual preparation of the board has to be performed. A detailed description how to prepare the board and how to install and use the software can be found in Nordic Semiconductor's user guide [Nor17b].

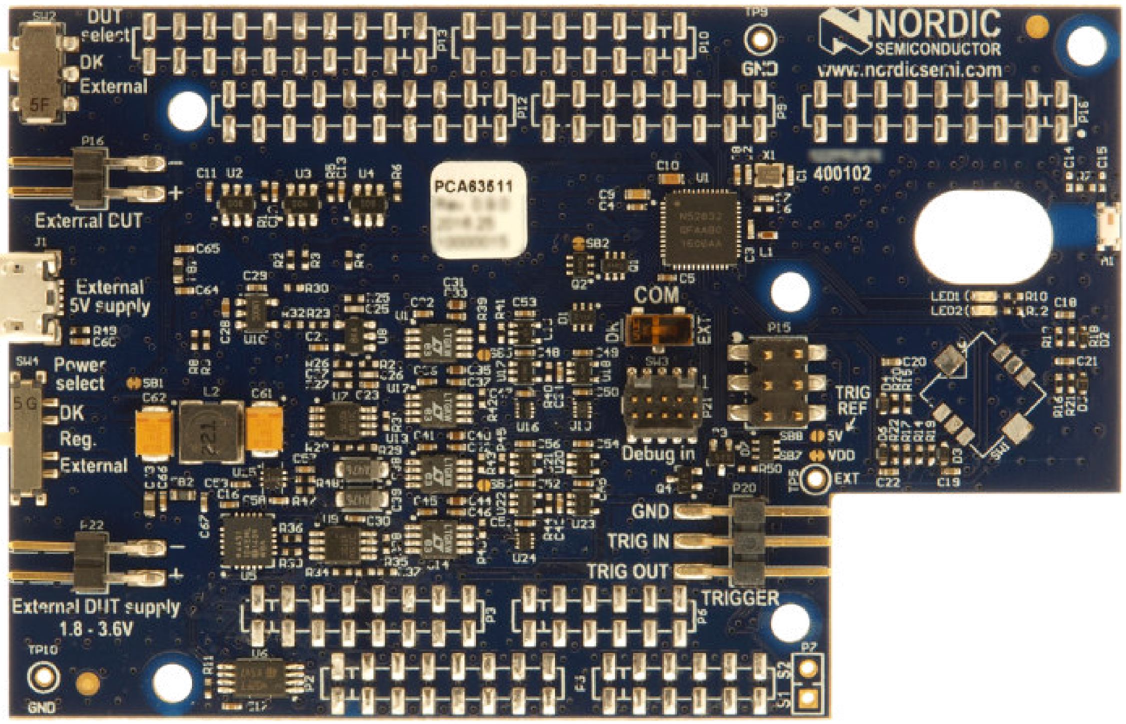


Figure 2.12: Nordic Semiconductor's Power Profiler Kit (nRF PPK) which is used in order to measure current consumption of the nDK.

Source: `http://infocenter.nordicsemi.com`

2.3 Zephyr Project

Zephyr Project [Fou17] is a small, scalable open source real-time operating system (RTOS) for embedded devices. We decided to use Zephyr as operating system for the nRF52 for several reasons:

- BLE v5.0 support for multiple hardware architectures: developing our project with Zephyr allows us to be flexible for the future. Zephyr already implements the new BLE v5.0 standard and supports many different hardware architectures. Therefore, we are not bound to Nordic devices and can change BLE chip afterwards.
- Open source: Zephyr is available through the Apache 2.0 open source license. It is totally free to use for commercial and noncommercial solutions.
- Modular: Zephyr offers a modular RTOS, optimized for memory and power constrained devices. It allows to enable/disable almost every feature (e.g., scan request notifications) over a config file in order to find the best solution for a specific use case.
- Adjustability: BLE parameters are adjustable with a fine granularity. Advanced features can be enabled when needed.

- Support: the operating system is under active development. It is possible to announce personal preferences and wishes that will be included in future releases.

The operating system is available for Linux, Windows, and macOS and can be downloaded directly from Github[3]. We tried out Linux and Windows and both were working as described on the official webpage. The Windows installation relies on MSYS2, which is a UNIX environment for Windows.

The setup description for Zephyr on the nRF52 can be found in Section 2.2.3.1.

[3]https://github.com/zephyrproject-rtos/zephyr

Chapter 3

Related Work

In this chapter we discuss the works that are related to this thesis. In Section 3.1 we list other smart locks available on the market working with different wireless technologies. Section 3.2 outlines modeling approaches for energy consumption and performance of BLE device discovery. Finally, in Section 3.3 we discuss existing runtime adaptation techniques of link layer parameters that have influence on device discovery, connection establishment, and connection management.

3.1 Smart Locks

As the NSL (see Section 2.2.1), most of the available smart locks use BLE to communicate with the user's smart phone or a special key fob. Wireless technologies such as WiFi, Z-Wave, or ZigBee are used to remote control or to include the smart lock into a smart home. Furthermore, most smart locks offer the possibility to keep track of the locking activities.

For example, the *Kwikset Kevo Bluetooth Deadbolt*[1] allows a user to unlock the door just by a finger tap on the lock. After the contact, the lock starts to search for the user's smart phone where the corresponding app is installed. Once the lock detected that the user is outside the door, it unlocks the door. As for the NSL, Kwikset provides a fob with which the lock can be used. Furthermore, the activity of all users engaging with a specific lock is tracked. In contrast to the NSL, the physical installation is harder, as the existing lock of a door has to be removed in order to mount the Kwikset Kevo on the door.

The smart lock *August*[2] is using a similar approach. Also in this case the existing lock has to be removed in order to be able to mount the smart lock. August is using BLE in order to communicate with the smart phone, auto unlocking when the user approaches, and auto locking when the user leaves. Every locking or unlocking activity is tracked with an activity log. Using the August WiFi-Bridge, the lock can be remotely controlled, similar to the NSL. Furthermore, an August door bell is available, including a camera that streams its captures to the user's smart phone. In addition to BLE and WiFi, August supports Z-Wave, making the lock easily capable of being integrated into a smart home.

[1]www.kwikset.com/kevo

[2]www.august.com

The smart lock *Danalock*[3] is an option once ZigBee or Z-Wave support is needed. Both versions are optional. In the basic version the smart lock can be used by BLE devices such as smart phones or tablets. As the NSL, the Danalock is not visible from outside. Still, the existing cylinder lock has to be replaced. Furthermore, all user interactions are tracked. Till March 2018 a WiFi-Bridge will be introduced allowing to remote control the smart lock. An additional feature of Danalock is the possible support of garage doors and gates by using an additional universal module.

The company *Noke*[4] offers a variety of BLE based smart locks in order to monitor shipping trucks and office doors, to secure dumpsters and employee lockers, and to lock a bike. With the phone application multiple locks can be managed, including an activity log and location history. Besides the smart phone also a key fob can be used to unlock the smart locks. Worth mentioning is the quick-click feature allowing to unlock a padlock without any device using a predefined code of short and long taps.

A further possibility to lock a bike is *Ellipse*[5], a smart lock based on BLE and equipped with a built-in solar panel that automatically charges its battery. It can be locked or unlocked using the smart phone. In case the phone runs out of battery a touchpad can be used to enter a predefined code. Furthermore, Ellipse is able to lock and unlock automatically using the received signal strength indicator (RSSI). As the lock can be located and access to it shared, Ellipse enables a smart bike share solution. Bike share members can use the mobile app to locate, book and access a bike.

The solutions presented in this thesis are not specific to the Nuki Smart Lock. Optimal Scan Parameters (OSP) can be used in any scenario where the scanner has knowledge about the advertising interval of the advertiser. The usage of Adaptive Advertising (AA) requires an activity log storing user interactions. Using this log, AA learns from user behavior and adapts the advertising interval of the device accordingly. Therefore, it can be applied on most smart locks and also on other devices with similar requirements. The idea of Range Extender (RE) can be used in any scenario two BLE devices are communicating with each other. Depending on the specific implementation, modifications of the design and implementation may be necessary.

3.2 BLE Modeling

Liu *et al.* [LCMX13] present a quantitative energy consumption analysis of BLE device discovery. They focus on the energy consumption of both advertiser and scanner, and outline the influence of their particular parameters (advertising interval, scan interval, scan window) on the energy consumption of each device.

Kamath *et al.* [KL10] show how to accurately measure the power consumption of a BLE peripheral (CC2541) while connected to another device, sending one or multiple data packets per connection event.

In contrast to the energy consumption, Jeon *et al.* [JDJ17] presented a model for analyzing the performance of BLE device discovery and outline the trade-off between device discovery latency and energy consumption. The efficiency of this trade-off is dependent

[3]www.smartlock.de
[4]www.noke.com
[5]www.lattis.io

on the parameter settings on both devices, advertiser and scanner.

Similar, Cho *et al.* [CPH$^+$14] developed an analytic model in order to evaluate the device discovery latency and probability in BLE networks. They showed that, with an increasing number of BLE devices, the device discovery latency may suffer under exponential growth, leading to the conclusion that there exist severe contentions among BLE devices. Those contentions are mainly caused by an inappropriate parameter selection.

In order to evaluate eligible parameters, Kindt *et al.* [KSC15] showed an algorithm that computes the discovery latency of purely interval based protocols such as BLE and ANT dependent on the selected parameters.

In this thesis, we model the energy consumption depending on the advertising interval and the advertising payload. We highlight the influence of the interaction of advertising and scanning parameters on the device discovery latency. Furthermore, we show that the device discovery latency can be minimized by choosing the advertising and scanning parameters appropriately.

3.3 Adapting Parameters at Runtime

Based on the performance analysis of the BLE device discovery process, Liu *et al.* [LCM12] show that BLE device discovery with standardized BLE parameters may lead to inefficient device discovery. As a solution the advertiser may adaptively reduce its advertising interval when encountering a long device discovery latency.

For Bluetooth ad-hoc networks, Drula *et al.* [DARD07] present adaptive policies for neighbor communication establishment. By dynamically changing the scan interval and scan window, they adapt the power usage of the discovery process depending on the probability of discovery success, which is calculated based on the location and recent activity.

Once a BLE connection is established successfully, Kindt *et al.* [KYGC15] show an adaptive online power management. Their approach allows a master device to save energy by dynamically adapting the connection interval dependent on the desired throughput.

In this thesis, we adapt the advertising interval of a BLE advertiser at runtime depending on user behavior. Therefore, an activity log storing past user interactions is necessary. Using this log, the device computes a daily schedule and adapts its advertising interval appropriately. This results in a reduction of energy consumption and device discovery latency.

Chapter 4

Modeling BLE Device Discovery

Before any action can take place, BLE devices have to perform device discovery, which includes advertising and scanning (see Section 2.1.2.1 and Section 2.1.2.2). Based on the distance and the selected parameters of both devices (such as advertising interval, advertising payload, scan interval, and scan window) the device discovery time and the consumed energy change. Therefore, in this chapter we want to gain a deep understanding of BLE device discovery.

Based on energy measurements, we develop a mathematical model to estimate the energy consumption of a BLE peripheral in Section 4.1. Furthermore, we present a model that estimates the energy consumption as a function of the advertising payload.

In Section 4.2, we investigate the influence of the transmission power on the line of sight (LOS) communication range. We show a simple model that allows us to estimate the maximum time for device discovery. This time is the period in which two BLE devices meet each other once one device is moving towards the other. This is of high relevance for use cases where a smart phone is involved. An example for such an use case would be a smart lock. A smart lock is advertising its presence and depending on its range we have limited amount of time, till the smart lock has to identify the user and unlock the door in order to avoid unnecessary waiting times.

In Section 4.3, we develop a mathematical model that gives the optimal scanning parameters (OSP) for a specific advertising interval in order to reduce the device discovery latency to a minimum. In order to prove the model, the device discovery latency is measured with different scan settings (scan interval, scan window).

4.1 BLE Advertisements

The energy consumption of a BLE peripheral is mostly determined by its advertising parameters. This includes the amount of channels the device is advertising on, the amount of advertised data, and the advertising interval. To maximize reliability under interference BLE devices should advertise on all three channels, and thus, we do not change this parameter. We model the energy consumption of the CDK based on the specific NSL use case (see Section 2.2.1) for different advertising intervals and for different advertising payload lengths.

Conversions from energy (E) to power (P) are realized with the standard power ($P =$

$U \cdot I$) and energy ($E = P \cdot t$) equations. Power is defined as the product of the supply voltage (U) and the current drain (I), which is measured using the Power Monitor (see Section 2.2.4). Energy is the product of power and time.

The starting point of the model is an energy consumption measurement with different advertising intervals over 60 s on the CDK. The measurement and its result is presented in the next section. The measurement does not take place on the NSL, as other energy consuming components that can not be switched off are running simultaneously and would thus influence the measurement in an unforeseeable way.

4.1.1 Measurements on the Cypress Development Kit (CDK)

We measure the energy consumption of the CDK for different advertising intervals and advertising payload lengths.

First, we measure the energy consumption on the CDK dependent on different advertising intervals. We measure the advertising intervals that are used by the NSL (see Section 2.2.1.2) and add a slower one that is used later in this thesis. Based on those measurements, we can calculate the mean power consumption for each advertising interval. In a further step, we use these values to estimate the energy consumption for longer periods (see Section 4.1.2). In order to simulate the NSL advertising as good as possible, the CDK imitates the advertising of the NSL. Therefore, the transmitted data is set according to the NSL advertising packet (30 bytes). The exact composition of the advertising packet is explained in Section 2.2.1.2.

Second, in order to estimate the energy consumption for different advertising payloads (see Section 4.1.3), we measure the energy consumption when in deep sleep (DS) mode.

Note that the NSL is also providing the local name consisting of 14 bytes, which is only sent as a scan response. This neither affects the energy consumption of one single advertising event, nor the energy consumption of the long term measurement, as it consists of advertising events only. This assumption holds as long as no other BLE device is scanning in the range of the advertiser. A power of 0dBm is used as transmission power.

For the measurement the Monsoon Solutions Inc. Power Monitor (see Section 2.2.4) was used. The power monitor measures the current drain, which can be measured on the CDK by removing the responsible jumper of the BLE chip and connect the VDD and GND pins to the power monitor. The Power Tool summarizes the total energy consumption.

The resulting values consist of the average of five measurements, each measured over a period of 60 seconds. The corresponding standard deviation was calculated.

T_{ADVI} [ms]	Mean Power [mW]
152.5	1.1615 ± 0.0030
417.5	0.4689 ± 0.0008
1022.5	0.2273 ± 0.0029
2000.0	0.1422 ± 0.0026
4000.0	0.0998 ± 0.0010
DS	0.0546 ± 0.0027

Table 4.1: Power consumption of the CDK, calculated based on the measured energy consumption over a period of 60 seconds.

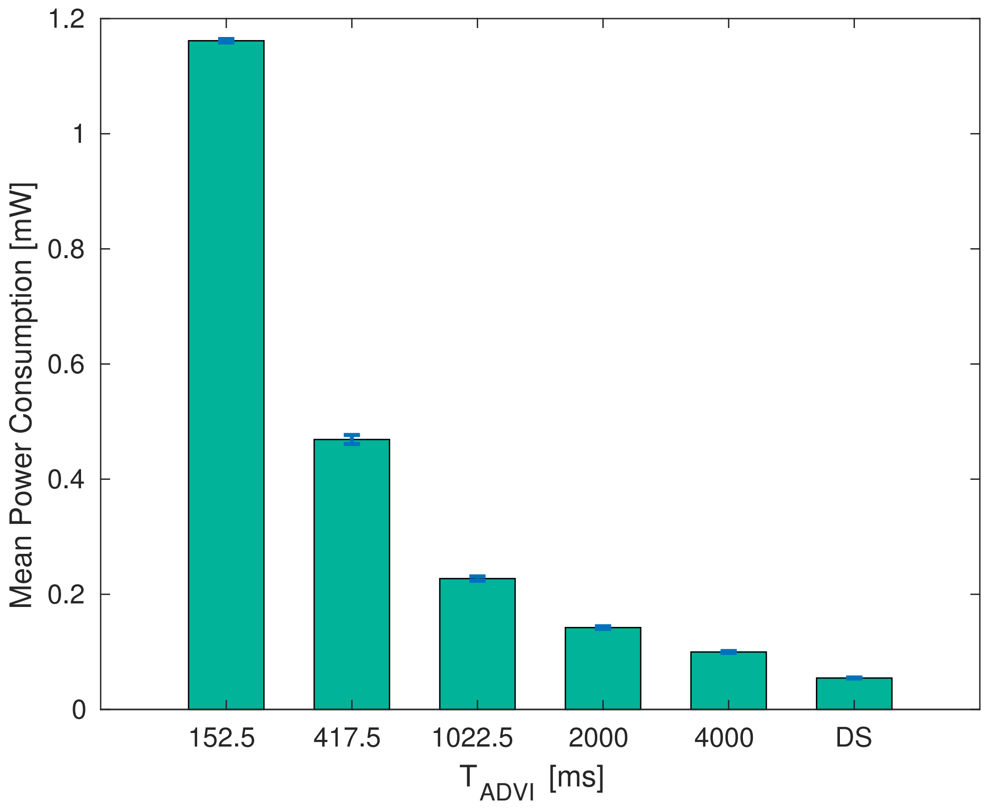


Figure 4.1: Mean power consumption of the CDK for different advertising intervals and being in deep sleep (DS) mode.

The results of the measurement are listed in Table 4.1 and are illustrated in Figure 4.1.

The measurement results are as expected: faster advertising intervals switch the radio on more frequently. Thus, the time spent in deep sleep is less and the power consumption is higher. Of course, despite the selected advertising interval, the power consumption of active advertising can never be less than pure deep sleep.

4.1.2 Mathematical Model

We derive a mathematical model in order to estimate the energy consumption of the BLE advertiser for longer periods and different advertising parameters. The power consumption dependent on a specific advertising interval was already calculated in Table 4.1. Based on those results, we can simply estimate the energy consumption of arbitrary long periods. The whole energy consumed during the considered period is E_{TOT}. This energy depends on the duration T of the period itself and the length of the advertising event T_{ADVE}, which consists of the duration of the advertising interval T_{ADVI} plus the random delay t_{Delay}. The random delay is a random value between 0 ms and 10 ms and is added at the end of each advertising interval in order to avoid persistent collisions with other advertising BLE

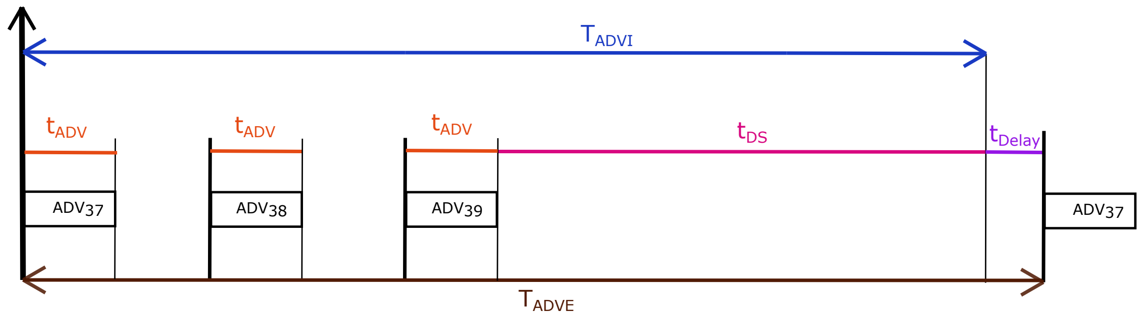


Figure 4.2: Progress of a BLE advertising event for the time T_{ADVE} using all three advertising channels.

devices. More details can be found in Section 2.1.2.1. The average energy consumed in one advertising event E_{ADVE} is the sum of the energy consumed in an advertising interval (E_{ADVI}) and the subsequent random delay (E_{Delay}). Furthermore, E_{ADVI} consists of the advertising energy E_{ADV}, consumed during three periods of t_{ADV}, and the energy E_{DS} of the time spent in deep sleep (t_{DS}). This relations are shown in Figure 4.2 and in Equation 4.1, where N_{ADVE} is the number of advertising events in the considered period.

$$\begin{aligned} E_{TOT} &= \frac{T}{T_{ADVE}} \cdot E_{ADVE} \\ &= \frac{T}{T_{ADVI} + t_{Delay}} \cdot (E_{ADVI} + E_{Delay}) \\ &= N_{ADVE} \cdot (E_{ADV} + E_{DS} + E_{Delay}) \\ &= N_{ADVE} \cdot (E_{ADV} + P_{DS} \cdot (t_{DS} + t_{Delay})) \end{aligned} \tag{4.1}$$

Note that Equation 4.1 assumes a uniformly distributed energy consumption over the whole period depending on E_{ADVE}. Of course, depending on the relative start of the first advertising event, it is possible that the last advertising event is not complete. To counter this, the start of the first advertising event has to be equal with the start T and N_{ADVE} has to be rounded down to the next natural number. From the fraction, the rest of the time in the period can be calculated. With the knowledge of how long each phase takes, we can add the consumed energy of all phases that still take place in the period T. This fact has a very small influence on relatively long periods T and is neglected in further calculations.

The exact value of the random delay is not known. As the delay is chosen between 0 ms and 10 ms, on average, we expect a random delay of $\bar{t}_{Delay} = 5ms$.

Equation 4.1 shows the importance of the power consumption in deep sleep mode. Depending on the selected advertising interval, a significantly high percentage of the whole time period is spent in deep sleep mode, which highly influences the overall energy consumption.

$t_{TX,Byte}$ [μs]	$P_{TX,Byte}$ [mW]	$E_{TX,Byte}$ [μJ]	$E_{TX,Byte,N_{ADVCH}=3}$ [μJ]
8	62.70	0.5016	1.5048

Table 4.2: Time, power and energy consumption for each transmitted byte.

4.1.3 Advertising Payload

In this section, the influence of the advertising payload on the energy consumption based on the presented mathematical model in Section 4.1.2 is elicited. The number of sent bytes has a direct impact on the deep-sleep phase, which makes a calculation of the consumed energy not trivial. The aim is to formulate an equation that expresses the energy consumption as a function of the payload.

An exact measurement of E_{ADV} needs expensive equipment. To avoid this, we adapt Equation 4.1 accordingly, allowing to estimate the energy consumption accurately, and managing without expensive equipment.

We know that, including overhead, the smallest advertising packet with no payload is 16 bytes. The maximum amount of payload is 31 bytes, which increases the size of the advertising packet to 47 bytes (see Equation 4.2).

In order to estimate the energy consumption based on the advertising payload, we have to think what each transmitted byte costs, meaning how long the radio has to be turned on for transmitting one single byte. In BLE the transmission time is given by Equation 4.3, where R_b is the Bitrate of 1Mbit/s. Applying Equation 4.3 leads to a transmission time of 8 μs per byte. The data sheet [Cyp17] states a current consumption of 16.5 mA while transmitting. In fact, the current consumption is higher as the whole system has to be awake. Measurements showed peaks of 19 mA during transmission. This leads to a power and energy consumption for one byte as stated in Table 4.2, based on a supply voltage of 3.3 V.

As we advertise on all three advertising channels, the time has to be multiplied by the factor of $N_{ADVCH} = 3$. This means that every byte more we send increases the energy consumption per byte by a factor of three (see $E_{TX,Byte,N_{ADVCH}=3}$ in Table 4.2).

$$N_{Bytes} = N_{Payload} + N_{Overhead} = N_{Payload} + 16 \tag{4.2}$$

$$t_{TX} = \frac{N_{Bytes} \cdot 8}{R_b} \tag{4.3}$$

Based on Equation 4.1, we know that every time unit not spent in the actual advertising time t_{ADV} is spent in deep sleep. This means that sending less bytes results in a shorter time needed for the advertising, which allows the device to go earlier to deep sleep. Vice versa, sending more bytes results in a shorter deep sleep phase and a longer time for advertising. The sum of both times stays constant in order to meet the advertising interval time T_{ADVI}.

Tripling the energy consumption per byte (see Table 4.2) leaves us with 1.5048 μJ consumed for each byte on all advertising channels. The energy for the same period of deep sleep is 1.3104 nJ. So the total extra power per byte can be expressed as the energy consumed per byte minus the energy consumed in DS during the same period of time.

y	x	calc. $E_{Byte} \cdot (y - x)$	t	calc. $E_{TOT,y}$	meas. $E_{TOT,y}$
[bytes]	[bytes]	[μJ]	[s]	[mJ]	[mJ]
30	30	0	60	-	69.6881
15	30	-22.3754	60	61.1631	56.9052
3	30	-40.2758	60	54.3429	51.4879

Table 4.3: Reducing the energy consumption by reducing the payload length of an advertising event from x = 30 bytes to y bytes: calculated change of energy consumption per advertising interval ($E_{Byte} \cdot (y - x)$), calculated energy consumption over a period of 60 s for y bytes (calc. $E_{TOT,y}$), and measured energy consumption over a period of 60 s for y bytes (meas. $E_{TOT,y}$).

y	x	Deviation
[bytes]	[bytes]	[%]
30	30	-
15	30	+7.48
3	30	+5.54

Table 4.4: Deviation of the calculated energy consumption compared to the measured energy consumption illustrated in Table 4.3.

Note that transmitting no data does not mean zero energy. During this time the device is in deep sleep mode. This results in an energy demand of **1.491696** μJ per byte sent (E_{Byte}) in a single advertising event when we advertise on all three advertising channels.

Equation 4.4 shows the resulting energy consumption over a considered period T, using an advertising interval T_{ADVI} compared to a previous energy value $E_{TOT,x}$, where x and y are the number of transmitted bytes for the corresponding energy value $E_{TOT,x}$ and $E_{TOT,y}$ respectively.

$$E_{TOT,y} = E_{TOT,x} + \frac{T}{T_{ADVI} + t_{Delay}} \cdot (E_{Byte} \cdot (y - x)) \tag{4.4}$$

Based on the previous measurement with a payload of 30 bytes, we estimate the energy consumption of the fastest used advertising interval (152.5 ms) based on two different amounts of payload (3 and 15 bytes). On the CDK a payload of 0 can not be selected, thus, the smallest possible payload is 3 bytes as the flag field (see 2.2.1.2) and the corresponding length has always to be sent.

Table 4.3 shows the result of the calculation (using Equation 4.4) and the measurement of the energy consumption based on different amount of payloads. According to the measurement, the reduction of the energy consumption is even more than estimated: the deviation of the calculation is illustrated in Table 4.4. The reason is probably in the previously made assumption of the current drain while transmission. We assumed a current drain of 19 mA. In fact, the current consumption is probably higher.

Nevertheless, we showed a valid mathematical model that demonstrates the considerable influence of the payload length on the energy consumption while advertising.

4.1.4 Outcome

Sending data in BLE advertising packets increases the energy consumption considerably. On the CDK, sending **27 bytes more** while using an advertising interval of 152.5 ms increases the energy consumption by more than **25%**. Thus, whenever possible try to reduce the amount of advertised data.

Remember, that BLE offers the scan response feature (see Section 2.1.2.2). This allows to transmit up to other 31 bytes of payload after a scanning device asked for it (scan request). Therefore, information that does not have to be inside the advertising packet could be moved to the scan response packet. Normally, scan requests occur much less frequent than the device is advertising. Thus, this can be a good way to save energy.

Unfortunately, for the use case NSL reducing the advertising payload is not possible. The NSL needs to represent an iBeacon (see Section 2.2.1.2), that needs 30 bytes of payload. Thus, no bytes can be elided.

4.2 Range

The usability of a BLE device depends on the discovery distance. This distance is the maximum distance at which device discovery is possible, i.e., the scanning device is able to receive advertising packets of the advertiser. Especially if we consider moving devices, this discovery distance is crucial. A larger discovery distance means more time is available once devices are moving towards each other in order to establish a connection or exchange data. In contrast to that, once they move away from each other, devices are able to transmit more data as they are for a longer time in transmission range.

In Section 4.2.1 we measure the line of sight (LOS) range of the CDK and the NSL. Based on those measurements, we present a simple model in order to estimate the maximum time for device discovery when one device is moving towards the other device (Section 4.2.2).

4.2.1 Range measurement

The NSL uses a transmission power of 0dBm. The maximum range, where a smart phone (Sony Xperia Z1 compact) was able to communicate over the NUKI App with the NSL in

P_{TX} [dBm]	$d_{RSSI=-87dBm}$ [m]
-18	1
-12	2
-6	3
-3	6
-2	8
-1	9
0	14
3	15

Table 4.5: Maximum LOS distance of the CDK for RSSI = -87dBm with different transmission power settings.

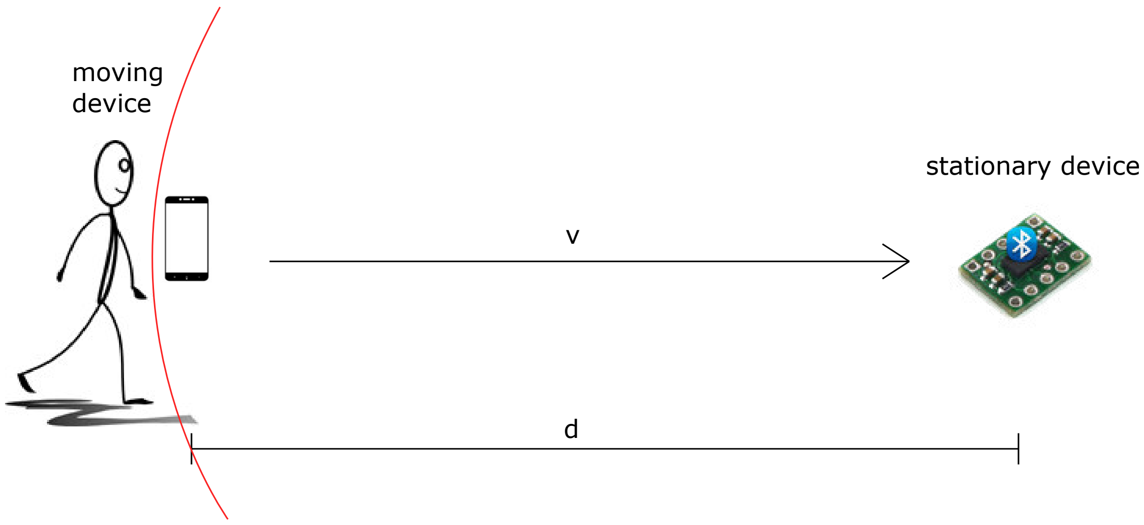


Figure 4.3: Maximum time for device discovery: time elapsing till the moving device with speed v at a distance of d reaches the stationary device.

the line of sight (LOS), was 14m: at this distance an RSSI of -87dBm was measured. For this measurement the successful reception of one single advertising packet is sufficient.

The same range measurement was performed with all power settings the Cypress BLE chip offers. To have comparable values with the NUKI Smart Lock, the distance at the RSSI of -87 dBm in the line of sight (LOS) was measured. The results are shown in Table 4.5.

Even though the BLE chip used in the NSL (which is the same as the CDK is using) supports a transmission power of 3 dBm, Nuki uses a transmission power of 0 dBm. The reason is that in this case doubling the transmission power does not pay off: using 3 dBm instead of 0 dBm gains one additional meter.

4.2.2 Maximum Time for Device Discovery

As maximum time for device discovery we define the time that a moving BLE device needs in order to reach a stationary BLE device. For simplicity, we assume that both devices are in line of sight (LOS). The stationary device is advertising and the moving device is scanning. The start of the time period is the time in which the moving device enters the range of the stationary device, meaning the scanning device is able to receive advertising packets sent by the advertiser. The moment in which the moving device reaches the stationary device using the shortest possible way terminates the time period.

We assume that the moving device moves with constant speed v towards the stationary device. The moving device gets in range of the stationary device at the distance d. The maximum time for device discovery t can be calculated by using Equation 4.5.

$$t = \frac{d}{v} \tag{4.5}$$

For example, using a LOS distance of **14 m** (transmission power of 0 dBm), measured in Section 4.2.1, and assuming that the moving device is approaching with a speed of **5**

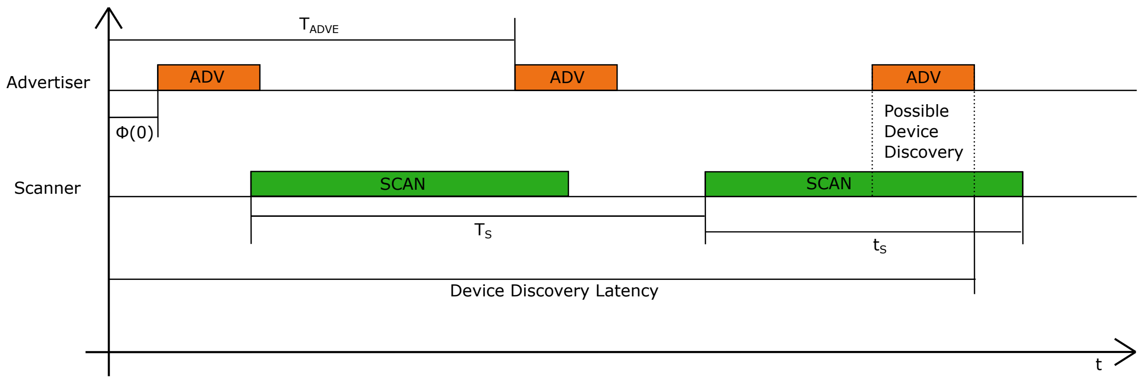


Figure 4.4: Device discovery latency dependent on the selected advertising event duration T_{ADVE}, scan interval T_S, and scan window t_S.

km/h (fast walking speed of a person) results in a maximum time for device discovery of **10.08 seconds**.

4.3 BLE Scanning

Usually, BLE device discovery takes place on three different channels (see Section 2.1.2). Device discovery is only possible if sender/receiver are using the same channel at the same time. Figure 4.4 illustrates the BLE device discovery process and the time spent for device discovery. The influence of the offset $\phi(0)$ is explained in the next section. Different configurations of advertising interval T_{ADVI}, scan interval T_S, and scan window t_S lead to a different device discovery latency and make its calculation challenging. In Section 4.3.1, we present a mathematical model in order to estimate the maximum device discovery latency under different advertising and scanning parameters.

Based on this model we introduce Optimal Scan Parameters (OSP) in Section 4.3.2. OSP consists of scan interval and scan window. By the knowledge of the used advertising interval of a remote device, OSP minimizes the amount of time spent for device discovery.

In Section 4.3.3 we evaluate the presented mathematical model by measuring the device discovery latency between two BLE devices depending on different advertising and scanning settings. We evolve the calculation of OSP by real measurements in comparison with the scan settings used by Android OS.

4.3.1 Mathematical Model

In order to determine the OSP, a mathematical model is developed. Some initial assumption have to be made:

1. No advertising packets are lost, i.e., we assume a 100% transmission reliability.
2. Fixed advertising intervals are used.
3. $T_{ADVE} = T_{ADVI} + t_{Delay} = T_{ADVI} + 10$ ms

The duration of one advertising event is the time of the advertising interval plus the random delay $\mathrm{t}_{\mathrm{Delay}}$ (see Section 2.1.2.1). This delay is a random value between 0 ms and 10 ms. We assume the maximum possible delay of 10 ms.

4. Depending on the used time axis (discrete or continuous), we distinguish between two different assumptions:

   - Discrete time axis: $T_{ADVE} < t_S$

     This ensures that, in each scan window, at least one advertising packet is received as each channel advertisement gets scanned at least once during one scan event.

   - Continuous time axis: $T_{ADVE} + t_{CH} < t_S$

     The start/end of a scan window can happen at every point of the advertising event. Adding the time spent advertising on each channel (t_{CH}) ensures that each scan window scans at least one channel successfully.

5. $(T_S$ - $t_S) \leq T_{ADVE}$

   For faster device discovery, the time in which no scanning is performed should be minimized. Therefore, we assume that the time without scanning in each scan interval is at least smaller than the duration of an advertising event. This prevents the partial reception of an advertising packet in one scan interval.

   To reach the fastest possible device discovery, continuous scanning ($T_S = t_S$) has to be performed.

In order to calculate the latency, a simplified model is introduced. Each advertising event is divided into four phases:

- A: advertising on channel 37
- B: advertising on channel 38
- C: advertising on channel 39
- D: phase with turned off radio in order to wait for the next advertising. This includes the random delay $\mathrm{t}_{\mathrm{Delay}}$. There we use the maximum possible delay of 10 ms (see Section 2.1.2.1).

This simple model allows us to deal with constant lengths of each phase.

The latency depends on which channel gets scanned at which point in time. In the worst case scenario, two other channels get scanned before the channel used by the advertiser. Therefore, the latency can be seen as number of needed advertising events that have taken place before the scan window where an advertising packet was received successfully, plus the advertising time of the channel itself. Equation 4.6 shows the calculation of the latency of the considered channel CH. $\phi_{CH}(0)$ describes the offset of the first advertising of the considered channel. t_{CH} represents the time spent while advertising on each channel. If the latency from the first advertising of the considered channel to its discovery of this channel has to be calculated then Equation 4.7 can be used. Thus, in this equation the offset $\phi(0)$ is elided. N_{SCANCH} is the number of channels that the scanner is scanning.

If this number is lower than the number of channels the advertiser is using (N_{ADVCH}), one or more channels can experience an infinite latency. If both are not using all three channels, then there is the danger that the device discovery will never be successful (e.g., advertiser is advertising only on channel 37, scanner is scanning only channel 38 and 39).

$$L_{max,CH} = T_{ADVE} \cdot \lceil \frac{T_S \cdot N_{SCANCH} - \phi_{CH}(0) - t_S}{T_{ADVE}} \rceil + t_{CH} + \phi_{CH}(0) \tag{4.6}$$

$$L_{max,CH,relative} = T_{ADVE} \cdot \lceil \frac{T_S \cdot N_{SCANCH} - \phi_{CH}(0) - t_S}{T_{ADVE}} \rceil + t_{CH} \tag{4.7}$$

4.3.2 Optimal Scan Parameters (OSP)

Based on the mathematical model presented in the previous section the OSP can be calculated: we derive a suited setting of scan parameters in order to minimize the device discovery latency according to a predefined advertising interval. In other words, we are looking for the maximum device discovery latency for optimal parameters. The calculation is illustrated with an example (see Figure 4.5). As we are interested in the maximum possible device discovery latency, we assume that the scan window of the considered channel is the third scan window (two other advertising channels may be scanned before). Once a channel is scanned for one time unit, its color changes to a shade of green. One device is always scanning, the other device is advertising: therefore, this example describes an unidirectional discovery.

We distinguish between two modes of scanning: non continuous and continuous scanning (see Section 2.1.2.2). Both modes are evaluated, illustrating their difference.

Non continuous scanning The first example presented in Figure 4.5 illustrates a non continuous scanning scenario. The numeric values are discrete time units, chosen in order to allow simple calculations, and are used for example only:

- $T_S = 8$
- $t_S = 5$
- $T_{ADVE} = 5$
- $t_{CH} = 1$

The maximum possible latency is achieved when the first scan window starts after $T_S - t_s$. Inserting the values above in equation 4.6 using the channels offsets ($\phi_A(0) = 3$, $\phi_B(0) = 4$, $\phi_C(0) = 0$) leads to the maximum latencies for each channel. This results in $L_{max,A} = 24$, $L_{max,B} = 20$, $L_{max,C} = 21$.

Continuous scanning Using continuous scanning results in the second example, illustrated in Figure 4.5. The numeric values are discrete time units, chosen in order to allow simple calculations, and are used for example only:

- $T_S = 6$
- $t_S = 6$

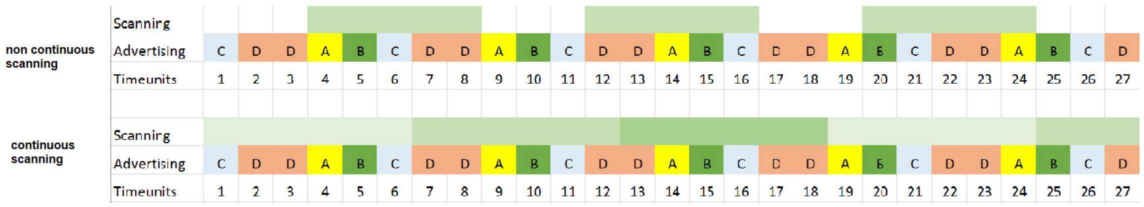


Figure 4.5: Device discovery latency calculation for non continuous and continuous scanning using a discrete time axis.

- $T_{ADVE} = 5$
- $t_{CH} = 1$

Using the same offsets as in the example for non continuous scanning, we calulate the maximum latencies $L_{max,A} = 14$, $L_{max,B} = 15$, $L_{max,C} = 16$.

The presented approach let us derive a suited setting of scan interval and scan window according to a predefined advertising interval in order to minimize the device discovery latency for BLE. The minimal latency is achieved when the following two points hold:

- $T_S = t_S$: use **continuous scanning**. This ensures that no advertising of a channel is missed.
- $(T_{ADVE} + t_{CH}) = t_S$: the scan interval is equal to the advertising event duration (advertising interval and t_{Delay}) plus the advertising time of one channel. This ensures the smallest possible scan window while **being able to scan at least one channel** successfully.

4.3.3 Evaluation

In order to evaluate the OSP calculations presented in Section 4.3.1, the device discovery latency is measured in a real world scenario with the three different scanning windows t_s (5000 ms, 2000 ms, 500 ms) supported by Android OS (see Section 2.1.2.4). Furthermore, the device discovery latency for a scan window that follows the OSP approach is measured. This means that, given an advertising interval, we add 10 ms for the maximum random delay t_{Delay} and the time of advertising on all three channels. This time depends on the amount of used advertising channels, the amount of bytes that have to be advertised and the inter channel transition time of the BLE chip. To be on the safe side, we have chosen 3.125 ms. Thus, in total, the OSP scan window is 13.125 ms more than the corresponding advertising interval. For an advertising interval of 4000 ms this would result in a scan interval and window of 4013.125 ms. With the measurement we demonstrate the influence of continuous/non-continuous scanning and the consequence of the length of the scan window on the latency.

4.3.3.1 Measurement Setup

For the measurement, the CDK is advertising with a fixed advertising interval T_{ADVI} (152.5 ms, 417.5 ms , 1022.5 ms , 2000 ms, and 4000 ms). The nDK executes a simple

T_{ADVI} [ms]	Scan Parameters			
	T_S [ms] / t_s [ms]	T_S [ms] / t_s [ms]	T_S [ms] / t_s [ms]	T_S [ms] / t_s [ms]
	5000/5000	5000/2000	5000/500	OSP/OSP
	latency [ms]	latency [ms]	latency [ms]	latency[ms]
4000.0	4145.8	6823.2	132580.0	3757.7
2000.0	1992.1	2222.1	57194.0	1986.0
1022.5	1078.7	1212.0	8512.0	1046.3
417.5	408.3	484.8	424.8	418.6
152.5	172.4	165.5	153.6	157.3

Table 4.6: Measured mean device discovery latency depending on the selected advertising interval (T_{ADVI}) on the CDK, as well as on the scan interval (T_S) and scan window (t_S) on the nDK.

program performing passive scanning. At the start of the scan, a timer is set to 0. The timer stops once the CDK was discovered. A successful device discovery consists of the successful reception of an advertising packet. After a successful discovery, the nDK stays inactive for a random period (0 up to 5 seconds). This step should simulate a real world application where devices can appear at any point in time. Subsequently, the scan process is started again. The measurement stops after 100 successful device discoveries. From all 100 measurement values the average is built. The resulting values are illustrated in Table 4.6.

4.3.3.2 Outcome

The measurement values presented in Table 4.6 allow several conclusions:

1. As long as the scan window t_S is larger than the selected advertising interval T_{ADVI}, the average device discovery latency remains close to the advertising interval value. This confirms our OSP assumptions. Selecting the scan window larger than the advertising interval assures that at least one advertising interval can be received during a scan window. Depending on the relation in time between start of scan window and start of advertising packet, the discovery time can be of different length. Note that we are using a random time of inactivity, which changes this relation from one measurement to the next one.

   Thus, even when selecting a shorter scan window, the average device discovery time remains close to the results of longer windows. This is because when starting the scan interval, the active part (the window) is executed first, followed by the inactive part of the scan interval. Once the first possible advertising packet is received without any errors, the average device discovery latency remains more or less the same. This behavior is illustrated in Figure 4.6a, where including the advertising interval of 2000 ms, the measurement averages of the three different scan windows (5000 ms, 2000 ms, OSP) are close to each other. The selection of a scan window of 2000 ms has an influence when the advertising interval increases to 4000 ms (green bar). There the device discovery latency increases dramatically.

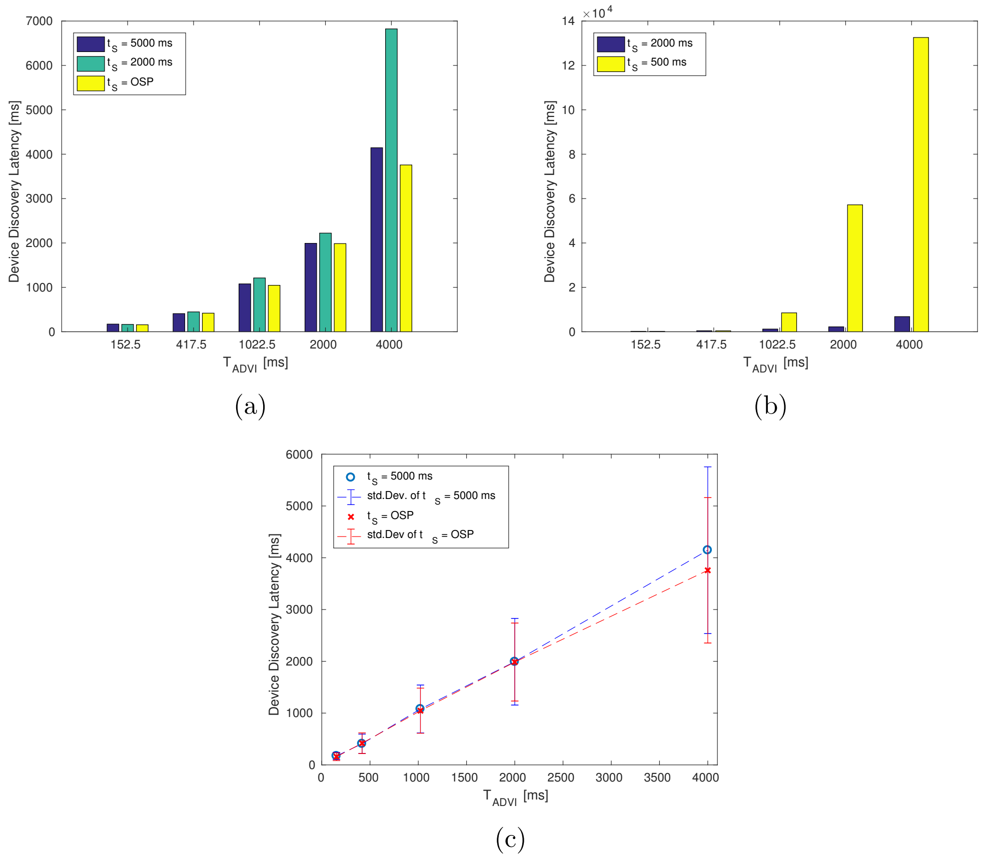


Figure 4.6: Device discovery latency dependent on advertising interval and scan window (a) (b) and the gain of OSP compared to a normal continuous scan window (c).

2. Once the advertising interval T_{ADVI} is longer than the used scan window, the device discovery latency increases to very high values. The reason is that now it is not assured anymore that at least one advertising packet can be received in a scan window. An unpredictable long time period can pass till a match in time between advertising packet and scan window occurs. This fact is illustrated in Figure 4.6b. Starting from the point where the advertising interval is longer than the scan window, the device discovery latency explodes. Of course, a bigger difference between advertising interval and scan window reinforces this effect.

3. The usage of OSP scanning policy pays off. On all measurements, independently from the selected advertising interval, OSP produced, with one exception, the best measurement values. The exception occurred with an advertising interval of 417.5 ms. There the latency value of a scan window of 5000 ms was slightly better. The reason for this behavior was explained in the first point: as long the advertising interval is shorter than the selected scan window, resulting values are close to the time of the advertising interval.

   The comparison between the device discovery latency of the two continuous scanning policies with a window of 5000 ms and OSP respectively is illustrated in Figure 4.6c.

As we can see, OSP performs slightly better. The calculated standard deviations are on a comparable level and show the influence of the randomization technique of appearing devices.

According to the measurement results presented in Table 4.6, we can see that on average, OSP reduces the device discovery time by **4.16%**.

We demonstrated that the device discovery latency depends on several factors such as **advertising interval, scan interval**, and **scan window**. What makes modeling complex is that the advertising and scan parameters have to be examined together. When developing BLE devices this fact has to be considered! We showed that OSP (selection of optimal scanning parameters as a function of the advertising parameters) is able to reduce the device discovery latency by more than **4%**. Thus, as a developer of one device (advertiser or scanner) we should have knowledge about the settings the other device is supporting and using. For example, it is not reasonable to choose a short scan window when we know that the advertising device is using long advertising intervals. In case advertising intervals are **not** changed over time, or other devices get informed in advanced about used advertising intervals, OSP can be of high relevance.

Further concepts presented in this thesis do not rely on OSP. Chapter 5 elaborates an advertising strategy, called Adaptive Advertising, which is independent from a scanning device. Adaptive Advertising is introduced in order to optimize device discovery from the point of view of an advertiser, minimizing latency and power consumption. In case of the strategy presented in Chapter 6, the Range Extender, the scanning device does not know the advertising interval used by the advertiser. Hence, for the Range Extender, the calculation and the usage of OSP is not possible.

Chapter 5

Adaptive Advertisement

Application requirements of BLE devices can change over time and can even be user dependent. As most BLE devices use static advertising intervals, those shifts in application requirements are not covered. This results in systems with a low energy efficiency or bad responsiveness.

The aim of Adaptive Advertising (AA) is to consider the constantly changing time, and user dependent application requirements. AA finds the right trade-off between energy consumption and responsiveness.

In Section 5.1, we describe the algorithm. We mention two different Adaptive Advertising strategies: Time of Day (TD) and User Behavior (UB). Furthermore, we highlight the possible savings regarding energy consumption.

Further sections are specific to Adaptive Advertising using User Behavior (AA-UB): in Section 5.2, we explain the design of the algorithm, followed by the implementation in Section 5.3. The algorithm can be fine tuned using several parameters. Their influence and the evaluation of AA-UB according to energy consumption and device discovery latency is explained in Section 5.4.

5.1 Algorithm

Adaptive Advertising (AA) means that the advertising interval T_{ADVI} is changed dependent on the current application requirements. This comes up with two advantages:

1. Responsiveness: device discovery latency can be improved by choosing a shorter T_{ADVI}.
2. Energy consumption: energy can be conserved by choosing a longer T_{ADVI}.

AA helps to find the right trade-off between device discovery latency and energy consumption. Among many other possibilities, we propose two different Adaptive Advertising strategies: Time of Day (TD) and User Behavior (UB). We focus on UB in our design and implementation.

5.1.1 Time of Day (TD)

Time of Day (TD) adapts the advertising interval according to a predefined advertising schedule, meaning the BLE device knows in advance at which point in time it has to use

$T_{152.5ms}$ [h]	$T_{417.5ms}$ [h]	$T_{1022.5ms}$ [h]	E_{TOT} [J]	Saving [%]
0	24	0	40.51	0
4	8	12	40.05	1.14
2	4	18	29.84	26.33

Table 5.1: Energy consumption based on different advertising intervals and savings with respect to the standard advertising interval of 417.5 ms.

which advertising interval.

With this approach, different advertising intervals can be used, allowing to target periods of different application requirements. We can say that the overall energy consumption for a time period T is the sum of the consumed energy in each period of a certain advertising interval T_i. Using M different advertising intervals and adapting Equation 4.1 this overall energy consumption $E_{TOT,adaptive}$ can be calculated as demonstrated in Equation 5.1.

$$E_{TOT,adaptive} = \sum_{i}^{M} \frac{T_i}{T_{ADVI,i}} \cdot E_{ADV,i} \tag{5.1}$$

Using this equation and the CDK energy consumption measurements illustrated in Table 4.1 we can estimate the energy consumption of the CDK when different advertising intervals are used. The result is illustrated in Table 5.1. This table illustrates the energy saving on the CDK over a whole day compared to a static advertising interval of 417.5 ms (first row), the default advertising interval of the NSL, dependent on how many hours a certain advertising interval is active. The second row of the table shows us that with the same amount of consumed energy, 4 hours per day can be spent in a faster advertising interval (152.5 ms) in order to increase the responsiveness significantly while energy is conserved during 12 hours per day. Those 12 hours could be spent during periods of inactivity, e.g., during night. The third row shows the potential of energy conserving for devices that do not need to be highly responsive over long periods. More than one quarter of energy could be conserved while reducing the responsiveness for 18 hours (1022.5 ms). Still during two hours per day a better responsiveness as with static advertising can be achieved.

TD is very simple and does not need any computation during runtime. Nevertheless, TD requires a priori knowledge about the application requirements, e.g., rarely used during night. An application could offer different advertising schedules that could be selected by users according to their own preferences. Of course, this would not represent an optimized advertising schedule, meaning that the time of high responsiveness may not be assigned appropriately. Therefore, we introduce User Behavior (UB), which we describe in the next section.

5.1.2 User Behavior (UB)

The strategy of TD presented in Section 5.1.1 can be improved by having knowledge about the specific usage of a device allowing to adapt the advertising schedule during runtime.

This means that UB matches the periods of high responsiveness with periods of high activity by using faster advertising intervals whenever needed and saving energy whenever it is possible. The knowledge about the usage of a device consists of several time stamps, created whenever the application was used (activity log) and stored directly on the specific device. Based on those past activities, UB generates a prediction for the future. It is not necessary that the data leaves the device. Those time stamps can be generated from many different users or other devices that interacted with the considered device.

Note that the estimated energy consumption presented in Table 5.1 holds also here. In contrast to TD, UB ensures that periods spent in faster advertising intervals are scheduled as efficiently as possible.

The design of the algorithm and its implementation is discussed in detail in Section 5.2 and Section 5.3, respectively.

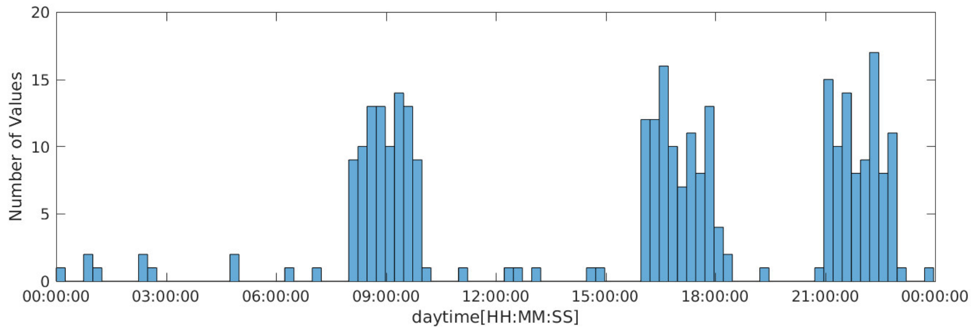
5.2 Design

Starting from the storage containing time stamps, we calculate the appropriate schedule on a daily basis. This means that after every day, new log entries are considered for the schedule of the consecutive day. The amount of entries is **not limited** by the algorithm, but dependent on the frequency. A storage of several hundred entries may be appropriate. For example, if we assume a storage of 300 values and ten entries per day, this would result in a time period of about one month. We decided to give all entries the **same weight**. This means that more recent entries have the same influence on the outcome as past entries. The aim is to design a highly variable schedule for the day that determines:

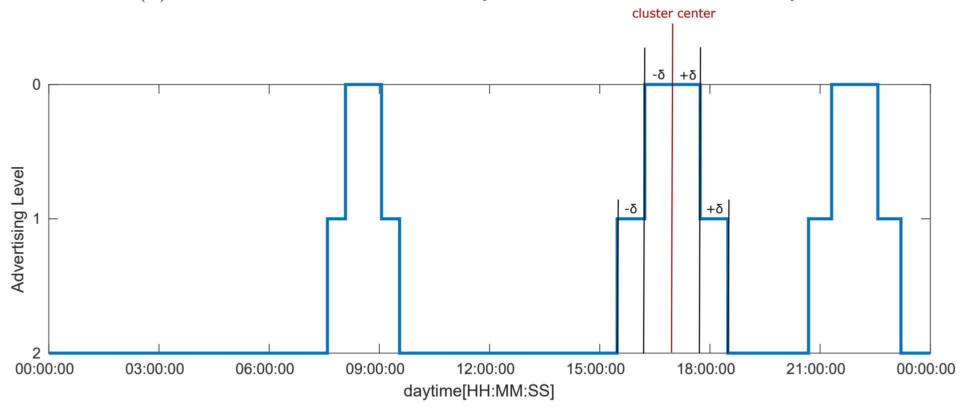
1. when/which advertising interval has to be set.
2. for how long an advertising interval has to be used.

All time values are mapped to a time line of 24 hours. Periods of high activity (many device discovery interactions) are sections with a high density of time values. As we want to match periods of high activity with periods of fast advertising intervals, we have to identify those agglomerations of time points. In order to find those clusters of time values, the algorithm **k-means** [Mac67] is used. This algorithm is used for vector quantization and cluster analysis. Points are categorized in k different clusters. We operate on a single time line but k-means can be used in a multidimensional space as well. The k-means algorithm adopted for our case is described in detail in Section 5.3.2.

Based on the resulting clusters of k-means and their standard deviation (δ), so called **points of interest (PoI)** are determined. The cluster represents the point with the highest density, meaning the fastest advertising interval that should be used around that point. The center plus/minus the standard deviation tells us the period in which the advertising interval should be used. Before and after the second fastest advertising interval is used. Both the start and the end of this period is a PoI. This calculation of PoIs continues till the slowest possible advertising interval is reached, resulting in a cascaded change of the current advertising interval. These are important points on the time axis, where an action (change of the advertising interval) *could* be necessary, as intersections with PoIs of other clusters may be possible. More information can be found in Section 5.3.3. Our algorithm decides which PoIs are relevant, depending on the selected parameters such as number



(a) Distribution of 300 activity values over the whole day.



(b) Advertising schedule over the whole day. Edges illustrate relevant PoIs.

Figure 5.1: Progress of advertising level during a single day for three different advertising levels and based on 300 activity values.

of supported advertising intervals, meaning on which points a change of the advertising interval *is* necessary. The number of supported advertising intervals is not limited. We speak about **advertising levels** (see Section 5.3.1), where level 0 is the fastest supported advertising interval. The outcome of the algorithm are several points on a daily time axis, telling the device that the current advertising interval has to be changed. An exemplary schedule is illustrated in Figure 5.1b, based on 300 activity entries. Furthermore, one of the cluster centers is marked with its PoIs (+/-δ). Their distribution over the whole day is illustrated in Figure 5.1a. Of course the profile depends on the current locking data and the selected params. The time points where an action (adaptation of advertising interval) has to take place and, therefore, the system has to be awake, are managed in the so called **wakeup array**. The calculation of the wakeup array is described in Section 5.3.4.

Applying the presented algorithm allows to accurately determine the time where a device should be in a highly responsive state or where it can advertise in a slower mode (e.g. during night). Based on this knowledge, the advertising interval can be adapted at runtime. This method does not increase the possible energy saving illustrated in Table 5.1, but ensures that the times in the different advertising modes are spent as efficiently as possible.

5.3 Implementation

The implementation of Adaptive Advertising (AA) was performed on the CY8CKIT-042-BLE Development Kit (CDK), that uses the CY8C4248LQI-BL583 BLE chip (see Section 2.2.2).

Periods where no actions take place allow to save a lot of power. This periods can be interrupted only by the execution of the AA-UB algorithm, including the calculation and the advertising itself. A user interaction always requires a previous advertising event. We do not have to care about waking up the system in case of advertising, as the BLE stack is handling this. As we want to eliminate static advertising, we need to define points in time, so called wakeup points, where the advertising interval is adapted according to a previously calculated schedule. The calculation of such a schedule and its realization is the core of the AA-UB algorithm (see Section 5.2).

The algorithm should run on constrained devices, which brings additional requirements: in order to ensure a predictable behavior, no variable data structures are used. Furthermore, all code is written in C in order to have full control over all structures and operations. The implementation is described in the following section.

5.3.1 Advertising Levels

The number of used advertising intervals is not limited and can be set through the parameter ADV_LEVEL. We speak about advertising levels, as any arbitrary advertising interval can be mapped to a certain advertising level. A higher ADV_LEVEL value means more supported advertising intervals and therefore more possible switches between advertising intervals may take place. This allows to use the algorithm for different preferences and use cases. It should be ensured, that level 0 is the fastest of the used advertising intervals. Higher level means slower advertising interval.

Note, that an ADV_LEVEL of 0 deactivates the adaptive advertising.

5.3.2 K-means

K-means is used to find agglomeration of time points, meaning periods with a high density of time values. The cluster center itself is a point on the time axis and the mid of the time that should be spent on advertising level 0. First, we have to define how many clusters we want to generate. The number is developer-defined by setting the parameter NUMBER_OF_CLUSTERS.

K-means is an iterative algorithm and after an initialization phase it consists of two steps:

1. Assign each point to its closest cluster.
2. Recalculate the cluster center by applying the arithmetic mean considering all points assigned to the cluster.

These two steps are repeated as long as the set (points) of a cluster changes with each iteration. The algorithm terminates if the set does not change anymore. Depending on the initialization of the cluster centers, the algorithm can take different numbers of steps. Furthermore, the initialization has influence on the final position of the cluster centers.

Usually, the cluster centers are initialized randomly, which can lead to a suboptimal position of the centers and an increased number of steps. To improve this, a simplified k-means++ initialization was chosen. Our approach initializes the first cluster center randomly from the existing data points and the next centers are selected based on their distance to all previous selected centers. The next center is the data point that shows the largest distance from all currently selected centers. The calculation takes more time than a random selection of the cluster centers, but ensures a fast convergence of the k-means algorithm. This lowers the computation time and therefore the energy consumption.

Once the cluster centers and all related points are calculated, the duration of each advertising level has to be determined. Therefore, for each cluster center, the standard deviation σ is calculated. Note that the standard deviation is cluster dependent and therefore differs from cluster to cluster! The value of the standard deviation is taken as width for each stage. The start of the first level 0 advertising is the center of the earliest cluster center minus the standard deviation of this cluster. The end of this stage is the cluster center plus the standard deviation. This results in a total width of 2σ. This calculation is valid for all other stages, which means that each advertising level has at most the duration of two times the standard deviation per cluster. The width of the different levels can be changed with the parameters **w** and **f** using the parameters W_DIV and F_DIV. The parameter w determines the width of advertising level 0, f determines the width of all other stages. If both define statements are 1, the standard deviation of the current cluster is used for w and f. Further explanation about their influence can be found in Section 5.4.2.2.

5.3.3 Points of Interest (PoI)

Once all clusters and their standard deviations are calculated, so called point of interest (PoI) can be determined. These points are important points on the time axis where an action (change of the advertising interval) *could* be necessary. PoIs are implemented as a struct containing four values:

- time: time value of the point.
- level: current advertising level.
- ignore: flag that allows to consider(0)/ignore(1) the point for further operations.
- edge: flag that defines the property of the transition of the current level. A falling edge (1→0) means an increment of the current level, a rising edge (0→1) decrements the current level.

For every cluster, an array with PoIs, dependent on the cluster center, ADV_LEVEL and the standard deviation σ, are calculated (see Figure 5.2).

If a high advertising level or number of clusters is selected, the calculated clusters are relatively close together. This can cause overlapping between time points of two or more clusters. To counter this, the so called points of interest (PoI), are prioritized. This prioritization follows four steps, starting from the most important one:

1. ignored-state: not ignored PoIs have higher priority

tc_0	tc_1	...	$tc_{NUMBER\_OF\_CLUSTERS-1}$	ADV_LEVEL = 0
tc_0 - w	tc_1 - w	...	$tc_{NUMBER\_OF\_CLUSTERS-1}$ - w	
tc_0 + w	tc_1 - w	...	$tc_{NUMBER\_OF\_CLUSTERS-1}$ + w	ADV_LEVEL = 1
tc_0 - w - f	tc_1 - w - f	...	$tc_{NUMBER\_OF\_CLUSTERS-1}$ - w - f	
tc_0 + w + f	tc_1 + w + f	...	$tc_{NUMBER\_OF\_CLUSTERS-1}$ + w + f	ADV_LEVEL = 2
tc_0 - w - 2f	tc_1 - w - 2f	...	$tc_{NUMBER\_OF\_CLUSTERS-1}$ - w - 2f	
tc_0 + w +2f	tc_1 + w + 2f	...	$tc_{NUMBER\_OF\_CLUSTERS-1}$+ w + 2f	ADV_LEVEL = 3
...	...	...	...	
tc_0 - w - (ADV_LEVEL - 1)f	tc_1 - w - (ADV_LEVEL - 1)f	...	$tc_{NUMBER\_OF\_CLUSTERS-1}$ - w - (ADV_LEVEL - 1)f	
tc_0 + w + (ADV_LEVEL - 1)f	tc_1 + w + (ADV_LEVEL - 1)f	...	$tc_{NUMBER\_OF\_CLUSTERS-1}$+ w + (ADV_LEVEL - 1)f	ADV_LEVEL

Figure 5.2: Structure of PoI array, depending on the number of clusters and the selected number of advertising levels. The width of level 0 is determined by w; f dictates the width of all other levels (see Section 5.3.2).

2. time: earlier PoIs have higher priority
3. level: lower level means a higher priority
4. cluster membership: if PoI belongs to a earlier cluster it has a higher priority

The prioritization is performed by the function *getHighestPriorityPoi()*.

5.3.4 Wakeup Array

Respecting the priorities, an array of PoIs is created. The resulting array is called wakeup array, as on each time point an action (increasing/decreasing the advertising interval) is needed. The function *calculatePoisAndWakeupArrayKmeans()* calculates the wakeup array, sorts it ascending, and returns the number of written time points. Several problems have to be solved here:

- **Influence of random time points**

  Consider a set of 300 data points without any weighting. The time values in a realistic scenario are not spread uniformly over the whole day. There are aggregations of points. As explained, k-means is used to determine those. The number of selected clusters determines the granularity: a smaller number of clusters leads to a larger standard deviation. Using a large number of clusters leads to clusters with only a small number of points. This so called random time points only occur a few times over the full period of 300 actions. If those points stick together, the damage is small. The cluster's standard deviation is small and, therefore, the time spent on fast advertising intervals is short. The energy consumption is increased slightly.

  This changes with a bigger distance between the different time points. If the distance is small enough so that no new cluster is created, but big enough so that the standard deviation increases significantly, the time spent on faster advertising intervals is increased. This increases the energy consumption as well. It was our intention to be highly reactive when it is necessary, but in this case there is no need to spend a lot of energy on a long time where then only a few locking actions are performed.

Therefore, a two staged **cluster weighting** was introduced, adjustable over two defines:

1. SECOND_LOWEST_LEVEL_ENABLE_MIN_PERCENT (SLLEMP)
   This statement allows to define a percentage threshold under that a cluster is ignored totally. For example, when selecting 5 as threshold, all clusters that do not contain at least 5% of all data points are ignored. Considering a number of 300 data points, all PoIs of clusters without at least 15 members are elided by setting the ignore flag accordingly. The consequence is a reduction of the current advertising interval to the slowest possible interval.
2. LOWEST_LEVEL_ENABLE_MIN_PERCENT (LLEMP)
   This statement allows to define a percentage threshold under that the fastest advertising interval (level 0) of a cluster is ignored. This value must be higher than SLLEMP. For example, when selecting 10 as threshold, all clusters with less members than 10% of all data points but more members than SLLEMP% are not ignored totally. The time spent on level 0 advertising (2σ) is downgraded to level 1 advertising. Therefore, for those clusters, no level 0 advertising is possible!

The two thresholds introduce a kind of cluster weighting. Clusters that have more members are more important as clusters with a number of entities smaller than the above explained thresholds. Energy can be saved by increasing the advertising level (slower advertising interval) of less important clusters.

- **Creating the wakeup array from an arbitrary number of PoI arrays**

  The number of clusters is variable. On the one hand, it depends on the selection of the parameter NUMBER_OF_CLUSTERS. On the other hand, the number of clusters can be reduced on runtime depending on SLLEMP. Furthermore, increasing ADV_LEVEL increases the number of PoIs per PoI array. This fact demands an algorithm that can deal with a large range of different numbers of clusters and advertising levels, in other words with a highly variable number of PoIs. Furthermore, PoIs can overlap and therefore overrule past/future decisions.

  To implement these requirements, prioritized insertion sort was used. The algorithm works iterative on all PoI arrays. In every iteration, the earliest PoIs (smallest time value) of all PoI arrays are selected and from all minima the highest priority PoI (see Section 5.3.3) is determined. Depending on the last inserted PoI in the wakeup array, the currently highest priority PoI is inserted or ignored. Note that in this stage, the decision of an insertion is dependent only from the previously inserted PoI! A PoI gets inserted when:

  1. The wakeup array is empty.
  2. The previously inserted PoI of the wakeup array has the same level but different edges, which is a switch from up to down or vice versa. But we have to take care: if both values, last inserted PoI and current PoI, have the same time value

two changes of the current level would happen simultaneously resulting in the previously selected advertising level. This is unnecessary. In this case, not only the current PoI is ignored but also the previously inserted PoI is removed.

3. The previously inserted PoI is of a lower level (higher priority) and the edges of both values, last inserted PoI and current PoI, are falling. This scenario describes a staircase-shaped reduction of the advertising interval.
4. The previously inserted PoI is of a higher level (lower priority) and the edges of both values, last inserted PoI and current PoI, are rising. This scenario is the opposite evolution as described previously: it is a staircase-shaped increment of the advertising interval. Here we have to consider two special cases: first, if the edges are not equal, the new PoI is overruling the old entry. Because of its higher priority, the new PoI overrides the previously inserted value of the wakeup array. Second, if both have falling edges, the new PoI of a higher priority overrides the previously written value.

The algorithm terminates when all PoIs of all clusters are inserted or ignored. The number of written values is saved.

- **Influence of PoIs over multiple wakeup array entries**

  So far, the decision if the current PoI is considered or ignored only depends on the previously inserted PoI. This could be a source of possible inconsistencies: besides the direct influence of the previously inserted PoI, the current PoI may depend on PoIs that are outside the closest neighborhood. On the one hand, these may be PoIs that are in a distance of multiple wakeup array entries but still follow a timely sequence. On the other hand, these may be PoIs that are "in the future", e.g., a high priority PoI shortly before midnight may overrule a lower priority PoI after midnight of the same day. This problem is solved by iterating over the wakeup array as many times as the wakeup array contains values. Some basic rules have to be met in each iteration in order to guarantee consistency:

  1. Every consecutive PoI has to be in a level range of one. If this does not apply, the lower priority PoI (higher level) gets ignored and removed from the wakeup array.
  2. Different edges of consecutive PoIs are only possible if they are of the same level. This can happen in two cases: first, when one advertising level starts at one point in time and ends later, where no other change is in between. Second, when one advertising level ends and starts at a later point in time again, while in the meantime no other changes occur.

     Else, the PoI with the smaller priority (higher level) gets ignored and removed from the wakeup array.

  Note that if we are talking about consecutive PoIs also the wrap around has to be considered. Therefore, those rules have to hold for the last and the first element of the array as well.

  Once the algorithm determined all relevant PoIs of the wakeup array, it returns the number of PoIs in the wakeup array. The wakeup array is sorted ascending.

- **Real Time Clock (RTC) and the alarm**

  To save as much energy as possible, all components that are not needed at the moment should be switched off. This includes the CPU as well. With the knowledge of the number of PoIs that are in the wakeup array, an easy wakeup mechanism can be implemented. This allows to switch off the CPU in the meantime and still be able to wakeup on variable, aperiodic time points. All we need is a Real Time Clock (RTC) and an alarm functionality. Luckily, the Cypress BLE Pioneer Kit with the CY8C4248LQI-BL583 BLE chip provides both functionalities. In order to be able to enter *deep-sleep* mode (see Section 2.2.2.2) a low frequency clock has to be used. Therefore we use the watch crystal oscillator (WCO) as source for the RTC. This can be configured in the Design Wide Resource File (.cdwr) under the clock tab inside PSoC creator. Once the RTC is initialised, we can set the first alarm to the time value of the first entry of the wakeup array. Once the alarm goes off, we set the new alarm to the time value of the next entry of the wakeup array. When the alarm with the time of the last entry of the wakeup array is reached, all wakeups of one days were performed and the next alarm is set to midnight. At midnight a recalculation of the wakeups, based on possible new locking actions take place. This includes the whole algorithm, containing k-means, standard deviation calculation, and wakeup array calculation. After the calculation finishes, a new alarm can be set, depending on the time value of the first entry of the wakeup array.

5.4 Evaluation

The evaluation of the AA-UB algorithm is based on the use case of the NSL, which is described in Section 5.4.1.

In Section 5.4.2 we analyze the behavior of the algorithm based on different data (simulation scenarios) and parameter settings. In Section 5.4.3 we measure the mean power consumption and the mean device discovery latency of the CDK (see Section 2.2.2). The results of both measurements are compared to the case where no AA is used (static advertising). Finally, we summarize the presented content in Section 5.4.4.

5.4.1 Use Case: NSL

The NSL stores the last **300** lock/unlock operations, categorized in three different types:

1. smart phone,
2. fob, and
3. manually

triggered lock/unlock operations. All three values disclose information about user activity and inactivity, but for us only the device triggered operations (1 and 2) are of interest. These tell us the exact time when the user established a connection to the NSL in order to lock/unlock the door. Advertising in general is only needed when a user wants to use the NSL, triggered by a device. A manual use of the NSL (without device discovery) is always possible. The task and the goal of AA-UB is to predict the user behavior in a way that the NSL is advertising when the user *is likely to* use the NSL and else minimize the rate of advertising. A perfect prediction is not possible, as unexpected events can occur at any point in time (*random time points*). In order to react to such events, advertising can never be turned off completely, else device-triggered operations do not work anymore. Therefore, we want to achieve low latency (fast advertising interval) on predict user interactions and saving energy consumption (slow advertising interval) whenever it is possible.

Lock/unlock entries can be generated from many different users. The NSL stores which user or which device (in case of the fob) interacts with itself. For us it is not important *which* user operates with the NSL. Therefore, all device generated entries have the same weight.

5.4.2 Simulation Scenarios and Parameter Settings

In this section, the algorithm is evaluated based on different time values (scenarios) and parameter settings according to the presented use case.

5.4.2.1 Simulation Scenarios

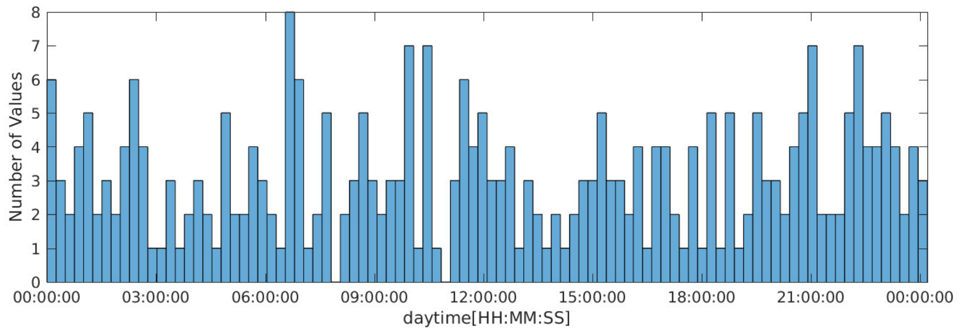
As we do not have access to any usage data of the NSL, we generate random time values for a smart lock. The values are generated using a Linear Feedback Shift Register (LFSR) as pseudo random number generator (polynomial 0xA3000000 in API Single Step mode). We distinguish between three scenarios: random time values, single household values and family household values.

The random scenario is used to evaluate the algorithm in the worst case scenario, i.e., no usage patterns are detectable. In contrast to that, the single household scenario evaluates the algorithm in case the smart lock is used by one person only. Finally, the family household scenario allows to evaluate the algorithm when multiple people are using the same lock.

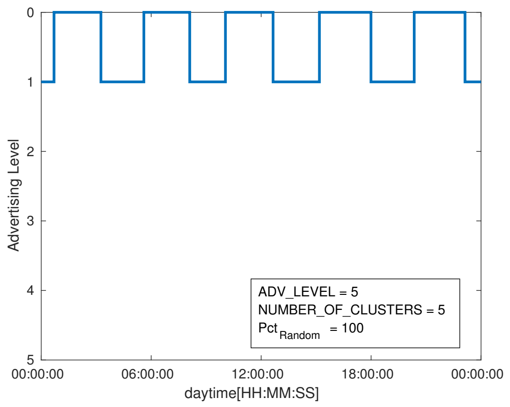
Random Time Values. In this case an array of only random time values, spread over the whole 24 hours, is generated (see Figure 5.3a). Therefore, the percentage of random values $\mathrm{Pct}_{\mathrm{Random}} = 100$.

This first scenario enables an investigation of the behavior of the algorithm in the **worst case** scenario: all time values are spread randomly over the full range. Projecting this scenario into the real world: a high number of persons is using the lock. Every person has a different locking behavior (different time when the lock is used) resulting in a 24h activity of the lock. Due to a non chronological weighting of the locking values, this scenario can also happen if only a few persons are using the lock and all of them showing scattered locking behavior over the maximum number of locking actions.

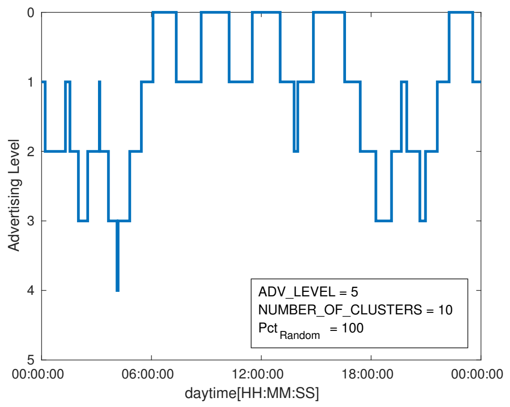
Figure 5.3 shows the performance of the developed algorithm based on random time values. Plots b-e are based on the same random values, illustrated in Figure 5.3a, in order to have comparable results. While the values of SLLEMP and LLEMP (see Section 5.3.4) remain constant, the setting of the used advertising levels and the possible number of clusters are changed. Putting the focus on Figure 5.3b shows that the number of possible clusters was fully utilized: we observe five different level 0 peaks. This means the fastest possible advertising interval is active once for every cluster. This is the outcome one



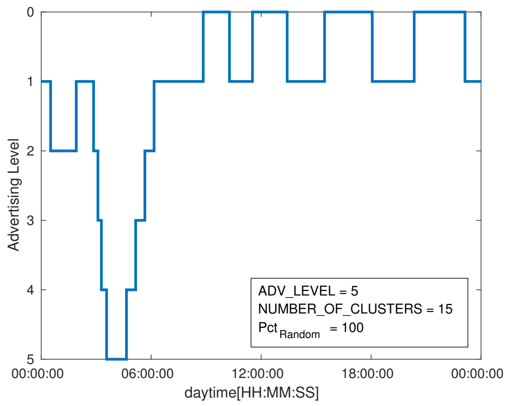
(a) Distribution of random time values over 24 hours.



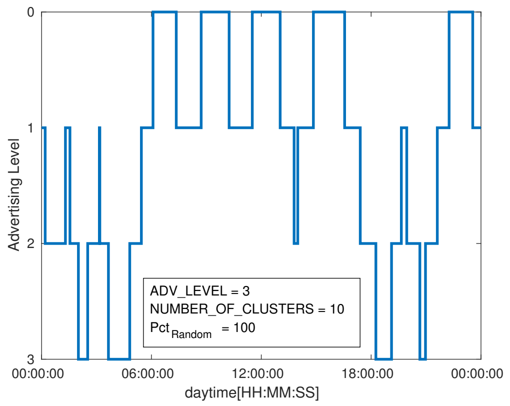
(b)



(c)



(d)



(e)

Figure 5.3: AA-UB algorithm applied on random time values (a) with different settings of the advertising level (ADV_LEVEL) and the possible number of clusters (NUMBER_OF_CLUSTERS). The values of SLLEMP and LLEMP (see Section 5.3.4) remain constant on 5 and 10 respectively.

would expect. The width of each peak is the same, which leads to the conclusion that the calculated standard deviation σ for every cluster is the same. The whole width of the peak is 2σ. The large σ prohibits an increment of the used advertising level, limiting the advertising levels from six possible to two that are actually used: level 0 and 1. This results in a slow latency but of course in a high energy consumption.

Figure 5.3c comes up with an interesting observation: doubling the number of clusters changes the plot significantly. The additional number of clusters introduces new possibilities (clusters) time points can be assigned to. Therefore, even in randomly distributed time points, agglomerations of points can be determined. Thanks to a smaller standard deviation, sliding down to slower advertising intervals (higher advertising levels) is possible. This reduces the time spent in lower levels and results in a significant energy reduction. Furthermore, we can see that the threshold of SHELMP is applied in order to save further energy. As not every cluster reaches the desired percentage of members, the lowest advertising level 0 is deactivated for those clusters. Therefore, some peaks stop at level 1.

Tripling the number of used clusters used in Figure 5.3b, leads to a similar outcome as we observed in Figure 5.3c. Thanks to the introduction of more clusters, the number of members per cluster is reduced. In this case, for some clusters the percentage of members is too small for the selected thresholds SHELMP. This leads to a total elimination of the cluster. Notwithstanding, we selected a higher number of clusters, the granularity is reduced as in the end less clusters are active as in Figure 5.3d. In this case, a reduction of the threshold SHELMP would be necessary in order to make use of all possible clusters.

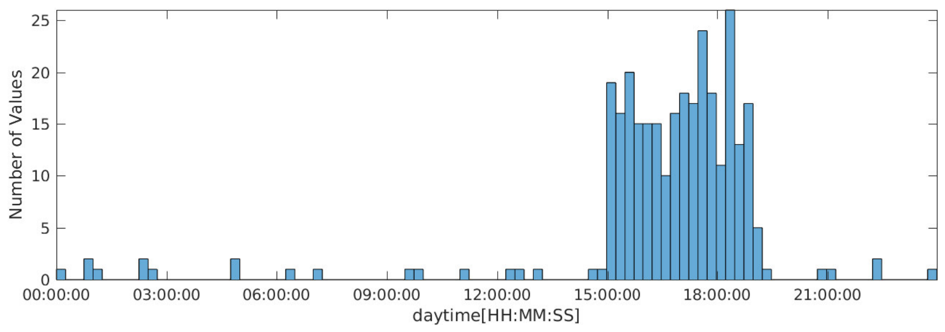
The impact of changing the number of used advertising levels is illustrated in Figure 5.3e. Comparing it with Figure 5.3c shows that the progression of the advertising level over the day is the same as one would cut off the Figure 5.3c horizontally at level 3. This leads to the conclusion that increasing the advertising level offers the possibility to return to slower advertising intervals on positions where the highest level is reached. In case of Figure 5.3c, the last possible level is not used. The plot with ADV_LEVEL 4 would be exactly the same. Therefore, having more advertising levels allows to save more energy whenever possible. Still it is not wise to choose a very high value as the computational effort increases rapidly. Furthermore, the device has to support the same number of different advertising intervals.

Single Household Values. For this scenario, an array of random time values is generated, where $\text{Pct}_{\text{Random}}\%$ of all values are spread over the whole 24 hours. The remaining 100% - $\text{Pct}_{\text{Random}}\%$ of all values are spread over the interval [t_{Start}, $\text{t}_{\text{Start}} + \text{T}_{\sigma}$]. An example of the distribution is shown in Figure 5.4a ($\text{Pct}_{\text{Random}} = 10$).

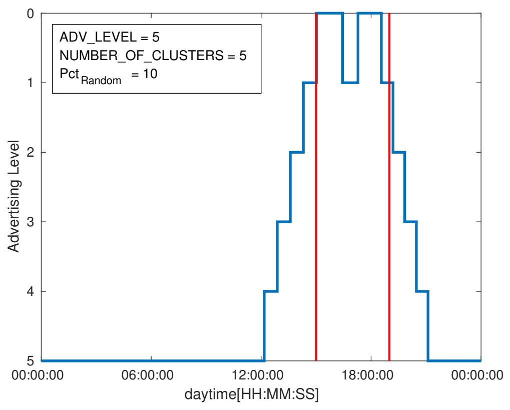
Using the generated values allows us to evaluate the algorithm in case the smart lock is used in a single household, i.e., there is only one person using the lock. This results in an agglomeration of the time values around a certain center time. For example, after returning every day from work at 5 pm most of the time values are spread around this value. Furthermore, this behavior can be achieved when multiple users are using the lock, having very similar locking behavior, meaning using the lock with a similar center time.

Figure 5.4 illustrates the behavior of the algorithm based on the discussed values. Comparing Figure 5.4b and Figure 5.4c shows the impact of the randomization.

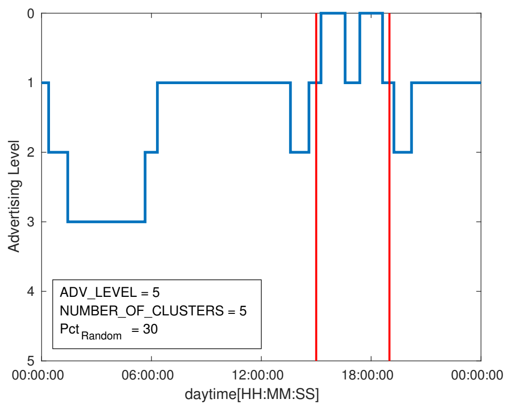
In the first example, a tenth of all values are outside the red boundaries. This is indicated by the percentage of random values $\text{Pct}_{\text{Random}}$. The remaining 90% of all values are randomly distributed inside the interval [t_{Start}, $\text{t}_{\text{Start}} + \text{T}_{\sigma}$], where t_{Start} = 15:00:00 and $\text{t}_{\text{Start}} + \text{T}_{\sigma}$ = 19:00:00. This example illustrates a person coming home every day



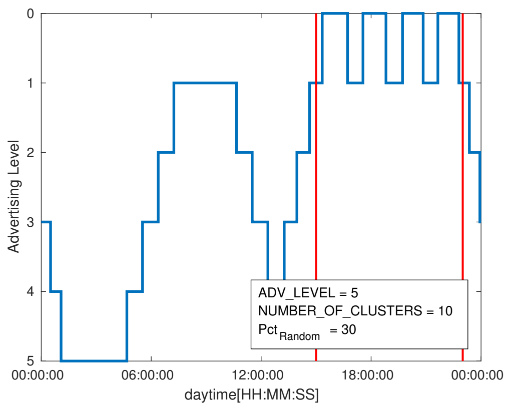
(a) Distribution of single household with a width of 4 hours and 10% of random values over 24 hours.



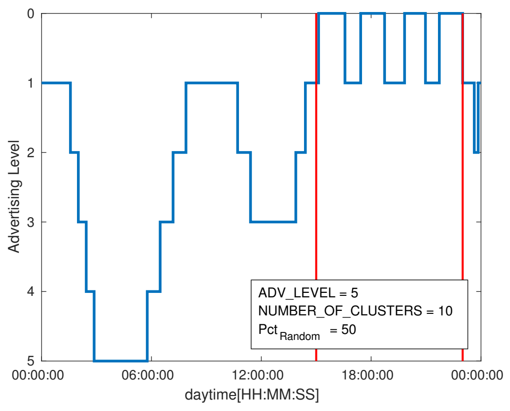
(b)



(c)



(d)



(e)

Figure 5.4: AA-UB algorithm applied on single household time values (a) with different possible numbers of clusters (NUMBER_OF_CLUSTERS) and different percentages of random values. The values of SLLEMP and LLEMP (see Section 5.3.4) remain constant on 5 and 10, respectively.

between 15:00:00 and 19:00:00 with only small anomalies. Therefore, we expect that the lock is using a fast advertising interval inside the red boundaries. As only very few points are outside (10%) and the threshold LLEMP was 10 itself, all clusters outside the boundaries do not have enough members. Therefore, they are elided. The device performs most of the time in the highest advertising level. This fact changes when we increase the percentage of random values that are located outside the selected boundaries to 30% (see Figure 5.4c). The threshold LLEMP is exceeded: the clusters outside the boundaries are not ignored. Still, there are not enough members per cluster in order to enable level 0 advertising outside the boundaries.

For the third example, the number of possible clusters is doubled in order to achieve a better granularity (see Figure 5.4d). Furthermore, the width of the boundaries T_σ is doubled. It ranges now from 15:00:00 to 23:00:00. This results in two consequences: first, more clusters are located inside the boundaries, allowing a finer setting of the advertising level. Second, the values outside the boundaries are spread over more clusters. This leads to the elimination of some clusters due to lack of members and therefore smaller advertising intervals can be used in order to save energy. Still, in time of the highest lock activity (inside the boundaries) the smart lock is operating at its two lowest levels. Outside the boundaries, thanks to SLLEMP, the lock is operating at levels higher or equal than 1.

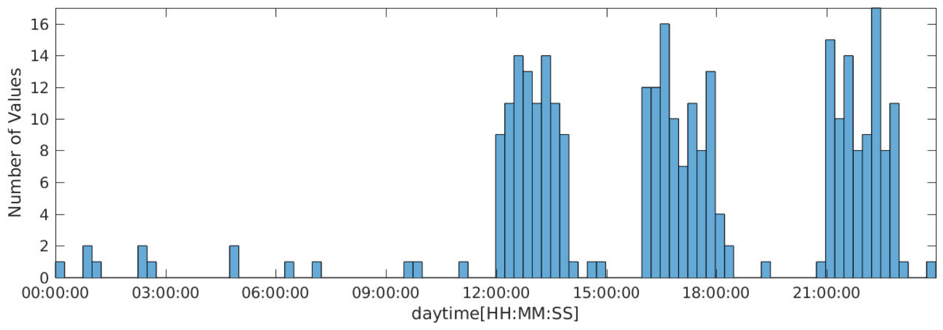
Going one step further and increasing the number of values outside the boundaries to 50% while keeping the other settings does not change the algorithms behavior inside the boundaries (see Figure 5.4e). Even with only one half of values remaining inside the boundaries, this is still the region with the highest density of time values. Therefore, most of the peaks are located there. Even with a high percentage of random values, the algorithm performs in a way where it enables high reactivity whenever it is needed (inside boundaries) and allows to save energy whenever its possible.

Family Household Values. Here, an array of random time values, where $\mathrm{Pct}_{\mathrm{Random}}\%$ of all values are spread over the whole 24 hours, is generated. The remaining 100% - $\mathrm{Pct}_{\mathrm{Random}}\%$ of all values are divided into three parts:

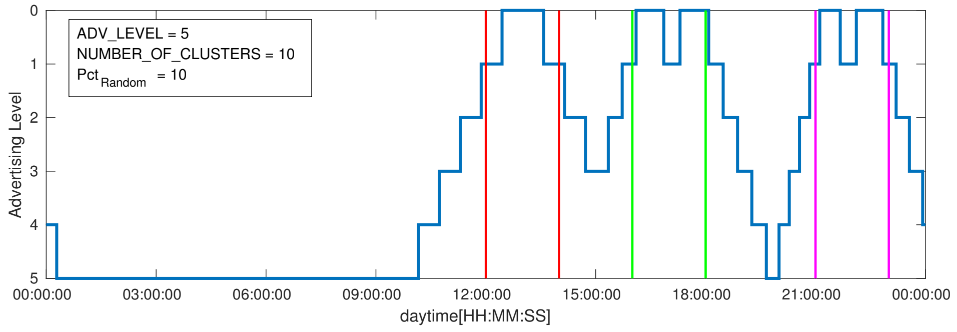
- $\frac{100\% - Pct_{\mathrm{Random}}}{3}\%$ of all values are spread over the interval $[\mathrm{t}_{\mathrm{Start},1}, \mathrm{t}_{\mathrm{Start},1} + \mathrm{T}_{\sigma,1}]$.
- $\frac{100\% - Pct_{\mathrm{Random}}}{3}\%$ of all values are spread over the interval $[\mathrm{t}_{\mathrm{Start},2}, \mathrm{t}_{\mathrm{Start},2} + \mathrm{T}_{\sigma,2}]$.
- $\frac{100\% - Pct_{\mathrm{Random}}}{3}\%$ of all values are spread over the interval $[\mathrm{t}_{\mathrm{Start},3}, \mathrm{t}_{\mathrm{Start},3} + \mathrm{T}_{\sigma,3}]$.

Using this set of values allows to determine the behavior of the algorithm when multiple people are using the smart lock resulting in three different main peaks. Figure 5.5b shows the resulting advertising level evolution. For this example, six different possible advertising levels and ten possible clusters are used. 90% of all values are located inside the three boundaries. More accurate, 30% of all values are located inside each boundary (see Figure 5.5a). As we can observe, all regions of high activity are hit by low advertising levels. Outside the boundaries, where power can be saved, the advertising interval is reduced.

In the second example (Figure 5.6b), the number of time values located outside the boundaries is increased to 40% (see Figure 5.6a). This means that two tenth of all values are located in each boundary. As explained before with Figure 5.4e, the width of each boundary is relatively small, therefore, the value density is higher inside each boundary. The algorithm determines the agglomeration of points and enables the fastest advertising interval. In comparison with Figure 5.5b, the calculated standard deviations are larger



(a)



(b)

Figure 5.5: Daily schedule (b) of AA-UB algorithm applied on family household values (a) with 10 clusters (NUMBER_OF_CLUSTERS) and 10% of random values (Pct_{Random}). The values of SLLEMP and LLEMP are set to 5 and 10, respectively.

resulting in broader pyramids. Furthermore, thanks to the higher number of random time values, an additional cluster appears outside the boundaries, but with not enough members to pass the LLEMP threshold. This illustrates a situation where the users of the smart lock return most of the time inside the presented boundaries of two hours (12:00:00 - 14:00:00, 16:00:00 - 18:00:00 and 21:00:00 - 23:00:00). In a real life scenario, this could be a normal week of work. There are some occasions like weekend, holidays or other users, who are using the smart lock, that create points outside the boundaries.

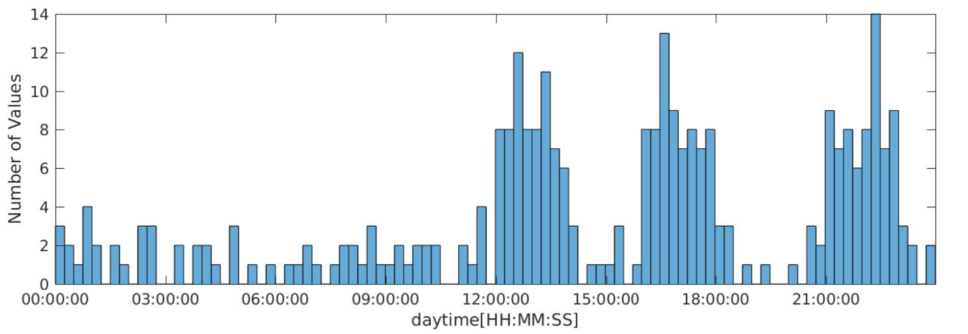
Imagine the situation where a company is using a smart lock (see Figure 5.7b).

1. Red boundary: the office hours are from 09:00:00 to 17:00:00. Workers arrive up to one hour earlier. For making lunch break, they need to leave the building. On their return they have to unlock the smart lock again. Suppliers who are allowed to use the lock can arrive during this time.

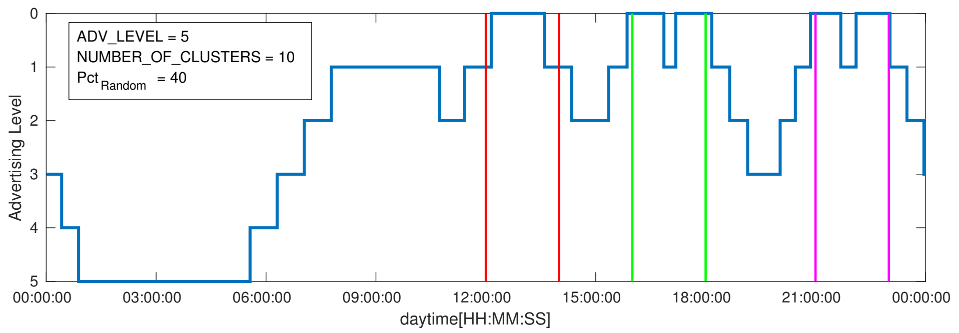
2. Green boundary: after the office closes, a team of workers handling night calls arrives.

3. Magenta boundary: the cleaning company accesses the building every night between 21:00:00 and 23:00:00.

Outside this daily five weekday schedule, there is a tenth of all locking actions. For simplicity, each boundary contains 30% of all time values. As we can see in Figure 5.7b,



(a)



(b)

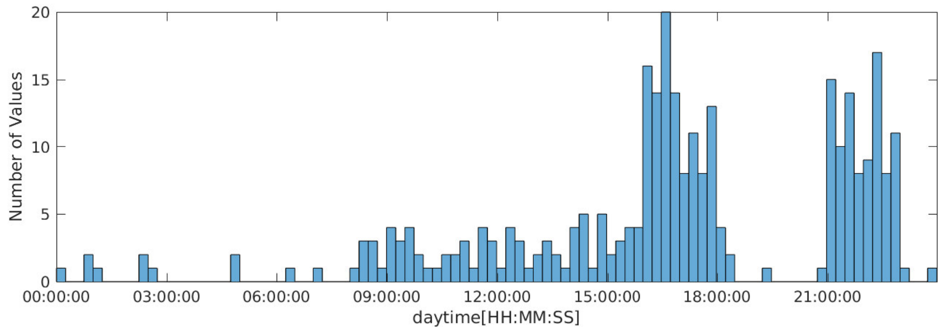
Figure 5.6: Daily schedule (b) of AA-UB algorithm applied on family household values (a) with 10 clusters (NUMBER_OF_CLUSTERS) and 40% of random values (Pct_{Random}). The values of SLLEMP and LLEMP are set to 5 and 10, respectively.

the device is in a lower advertising level inside the boundaries. This ensures a better user experience: lower advertising level means a faster active advertising interval which reduces the expected latency when a locking actions has to be performed. During night, where only a tenth of all locking actions are performed, a lot of energy can be saved using higher advertising levels.

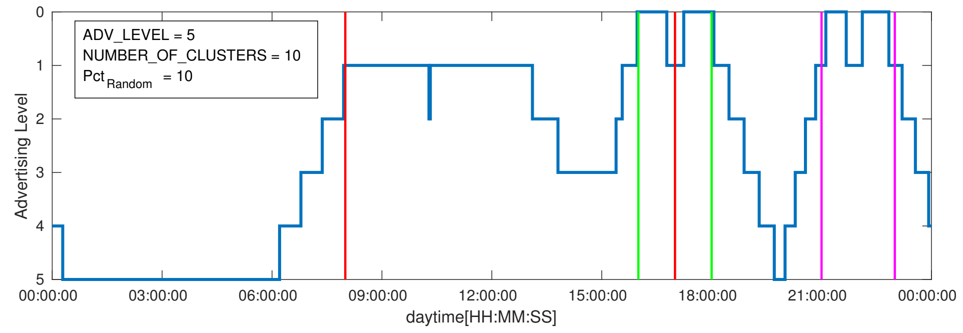
Applying the algorithm with an increased number of possible clusters to the same data as explained with Figure 5.7b results in Figure 5.7c. As already outlined in the explanation of Figure 5.3, an increased number of clusters allows a finer distinction of the time values. This is an important property when inside an agglomeration several smaller agglomerations are located, i.e., a boundary can be split into smaller boundaries.

5.4.2.2 Parameter Settings

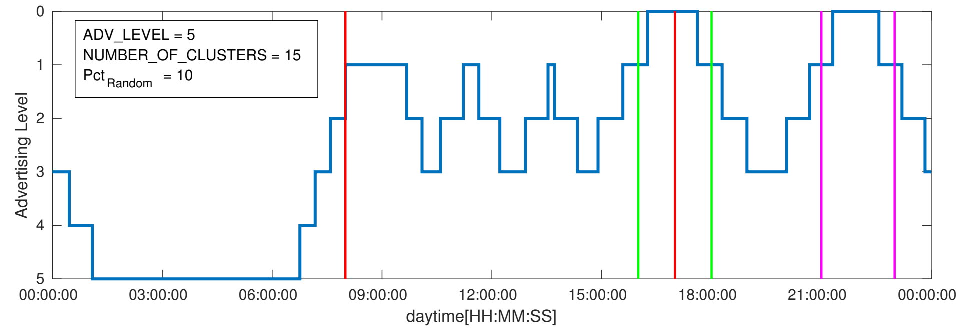
In this section, the influence of the several adjustable parameters are shown. ADV_LEVEL selects the possible number of advertising levels. Using NUMBER_OF_CLUSTERS allows to select the maximum number of clusters. W_DIV and F_DIV control the width of the different stages. More precisely, the parameter W_DIV determines the width of advertising level 0, F_DIV determines the width of all other stages. LLEMP defines a percentage threshold under that advertising level 0 of a cluster is ignored. In contrast to that,



(a)



(b)



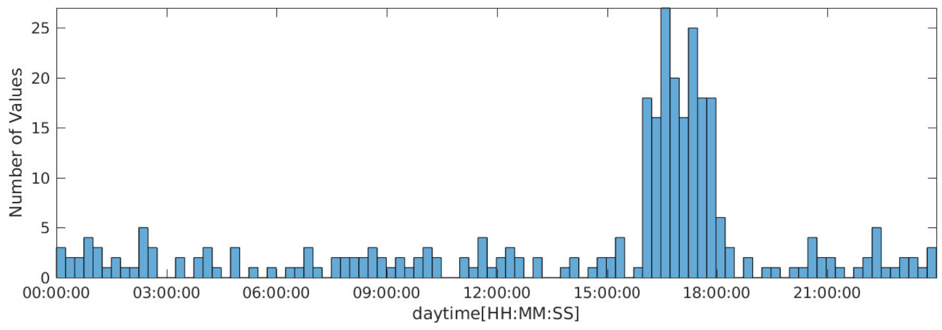
(c)

Figure 5.7: Daily schedule (b, c) of AA-UB algorithm applied on family household values (a) with different number of clusters (NUMBER_OF_CLUSTERS) and 10% of random values (Pct_{Random}). The values of SLLEMP and LLEMP are set to 5 and 10, respectively.

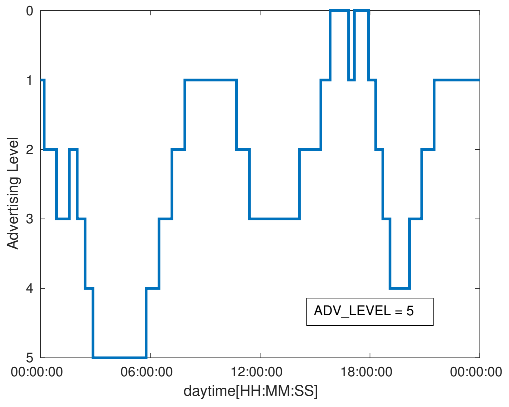
SLLEMP defines a percentage threshold under that a cluster is ignored totally.

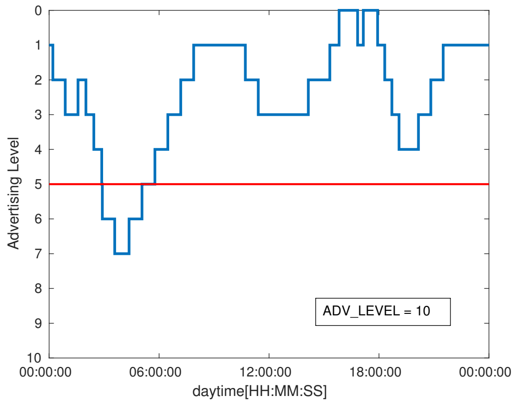
ADV_LEVEL With this parameter the number of possible advertising levels can be selected. There are ADV_LEVEL + 1 different advertising levels.

Figure 5.8 shows the influence of the parameter ADV_LEVEL. Figure 5.8a illustrates the generated values. In Figure 5.8b an advertising level of five was chosen. In contrast to that, in Figure 5.8c the advertising level was doubled. As demonstrated by the figures, a higher advertising level offers more possible advertising intervals but does not guarantee



(a)





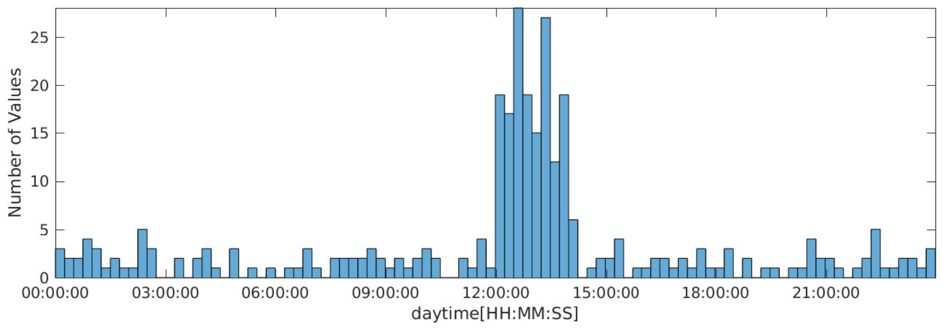
(b) (c)

Figure 5.8: AA-UB algorithm applied on the same data (a) with varying ADV_LEVEL parameter (b, c).

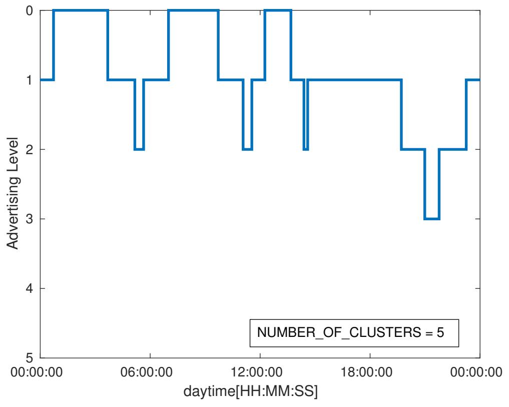
that all of them are used, e.g., in the figure on the right side only the levels 0-7 are applied. If we take a closer look, above the red line the two plots are identical. Increasing the ADV_LEVEL adds further possible reductions of the advertising interval once the highest advertising level is reached.

NUMBER_OF_CLUSTERS With this parameters, the maximum possible number of clusters can be selected. This does not guarantee that all clusters are active! Depending on the thresholds LLEMP and SLLEMP clusters can be partially or totally ignored.

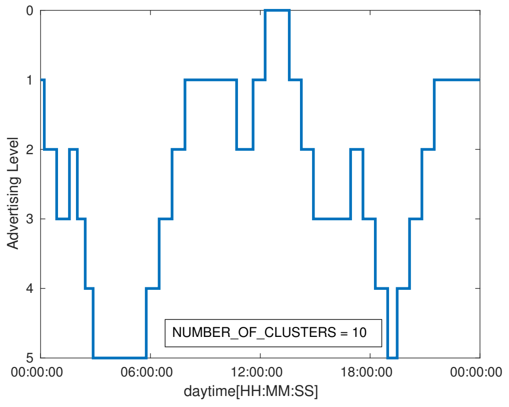
Based on the values illustrated in Figure 5.9a and in comparison with Figure 5.9b, Figure 5.9c shows that increasing the number of clusters increases the granularity. In general calculated standard deviations getting smaller leading to a finer trend of the advertising level. This holds as long clusters remain active. If values are spread randomly over the whole day, it can occur that even with higher cluster number the output gets flatter, i.e., the algorithm does not make use of the high advertising levels. So in addition to the energy spent on the computation, more energy is consumed as the higher levels are not used. Therefore, increasing the number of clusters should happen with care!



(a)



(b)



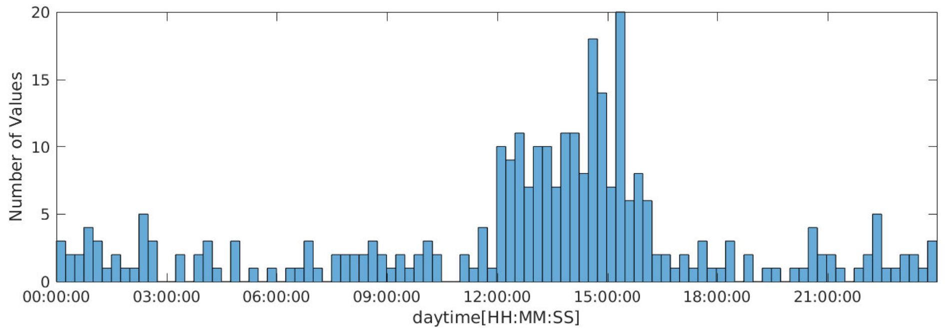
(c)

Figure 5.9: AA-UB algorithm applied on the same data (a) with varying NUMBER_OF_CLUSTER parameter (b, c).

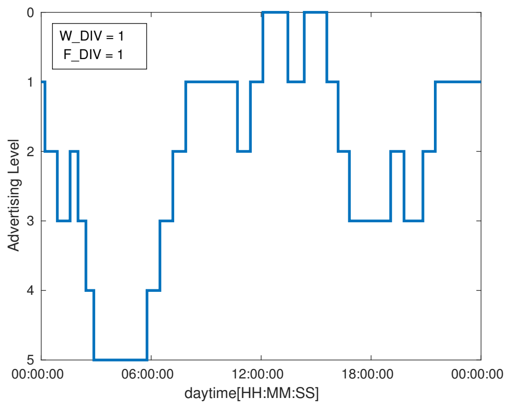
W_DIV and F_DIV The width of the different stages can be controlled with the paramaters W_DIV and F_DIV. W_DIV is operating as divider of the width of the advertising level 0. If set to 1, no divider is active and therefore the width of the advertising level 0 is two times the standard deviation of the cluster. F_DIV is the divider of the width of each consecutive stage (see Section 5.3.2 and Figure 5.2). If set to 1, the width of each stage is at most 2σ. Again, a higher F_DIV allows to reduce this width. Remember that both dividers are used for an integer division. Therefore, choose this values with care.

Figure 5.10 shows the outcome when different W_DIV and F_DIV are used. Both plots are based on the same data points (see Figure 5.10a). Figure 5.10b shows the result of the algorithm when W_DIV and F_DIV are set to one. Therefore the width of the level 0 peaks are 2σ. On the peaks where level 0 advertising is deactivated thanks to LLEMP, the width is at most 4σ. Remember that f is 2σ per stage if no divider is applied. For all other stages, the width on each side of the pyramid is at most σ.

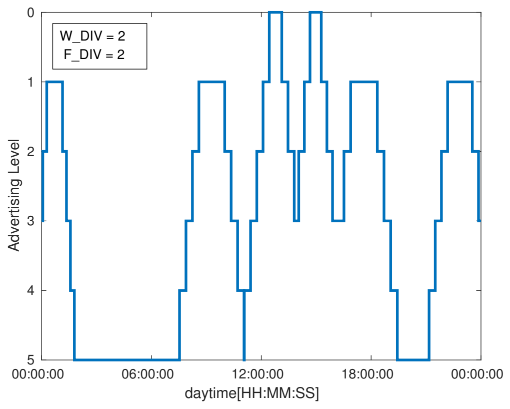
In Figure 5.10c the dividers for w and f are doubled, i.e., w and f are halved. As we can observe, the number and position of peaks does not change but the width of every stage is halved. Spending less time on each level opens the possibility to make use of higher levels in between of each peak. This means that selecting higher dividers saves more energy, but has a negative impact on the latency.



(a)



(b)

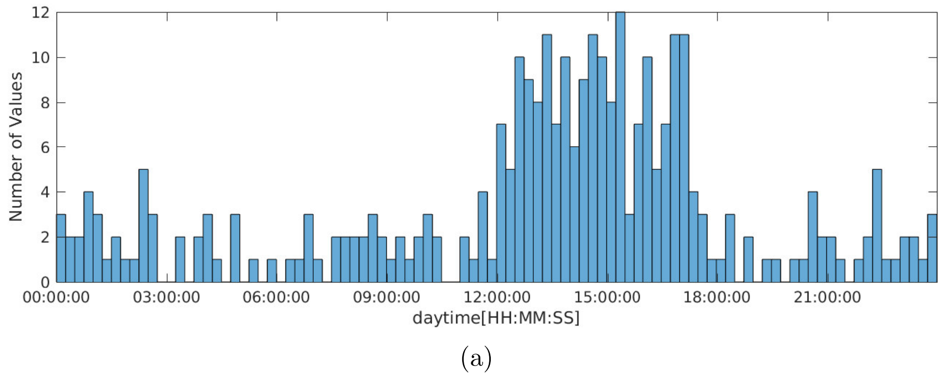


(c)

Figure 5.10: AA-UB algorithm applied on the same data (a) with varying parameters W_DIV and F_DIV (b, c).

LLEMP and SLLEMP Thanks to the threshold SECOND_LOWEST_LEVEL_ENABLE_MIN_PERCENT (SLLEMP) clusters with not at least the stated percentage of members are ignored totally. This allows to counter the creation of a full advertising level pyramid for only a few values. As we can see in Figure 5.11b, only three clusters reached level 0. Remember that once a cluster is accepted all advertising levels are passed if they are not overruled by lower levels. For further explanation see Section 5.3.4. The only exception can be achieved with the threshold LOWEST_LEVEL_ENABLE_MIN_PERCENT (LLEMP). Using this threshold, a percentage of cluster members can be defined above which the full advertising pyramid is built (level 0 - ADV_LEVEL). If a cluster does not exceed this threshold, the advertising level 0 is disabled. In that case, the lowest possible level is level 1 (see Figure 5.11). LLEMP should be higher than SLLEMP. Figure 5.11c shows the plot of the advertising level trend on the same data (see Figure 5.11a) as in Figure 5.11b but with halved LLEMP and SLLEMP parameters. We observe two things: first, more level 0 peaks are present. This means more clusters were able to exceed the LLEMP threshold. Second, the highest level that is reached is level 3. This implies that in comparison to the left figure, more clusters are present, because they were able to pass the SLLEMP threshold.

Therefore, using higher thresholds reduces the number of active clusters and enables



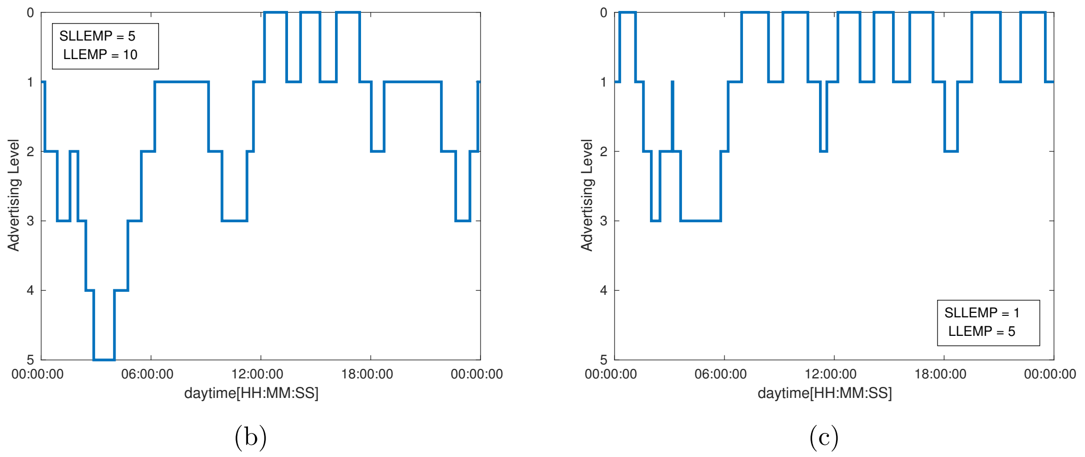


Figure 5.11: AA-UB algorithm applied on the same data (a) with varying parameters LLEMP and SLLEMP (b, c).

a higher energy saving. As consequence, the latency is increased. Note that reducing NUMBER_OF_CLUSTERS does not lead to the same result: there is a difference between the number of possible clusters (NUMBER_OF_CLUSTERS) and the number of active clusters, that is calculated by the algorithm!

5.4.3 Power Consumption and Device Discovery Latency

In this section, the power consumption and the resulting device discovery latency of the algorithm is estimated, measured, and evaluated according to the three different simulation scenarios (random, single household, and family household) presented in Section 5.4.2.1. We compare the results with the measurements of static advertising in order to highlight the gain of AA. All power measurements were performed on the CDK (see Section 2.2.2) using the power monitor (see Section 2.2.4).

For the evaluation of this section five different advertising levels were chosen. The levels and their assigned advertising intervals (T_{ADVI}) are illustrated in Table 5.2.

First of all, we need to measure the mean power consumption and the mean device discovery latency of the CDK while performing static advertising, based on the five different advertising intervals. The results of the measurements are already listed in Section

Advertising Level	T_{ADVI} [ms]
0	152.5
1	417.5
2	1022.5
3	2000.0
4	4000.0

Table 5.2: Advertising levels and their assigned advertising intervals ($\mathrm{T_{ADVI}}$) used for the power consumption measurement on the CDK.

T_{ADVI} [ms]	Mean Power [mW]	Latency [ms]
152.5	1.1615	172.4
417.5	0.4689	408.3
1022.5	0.2273	1078.7
2000	0.1422	1992.1
4000	0.0998	4145.8
DS	0.0546	-

Table 5.3: Mean power consumption and device discovery latency of the CDK dependent on the used advertising interval ($\mathrm{T_{ADVI}}$).

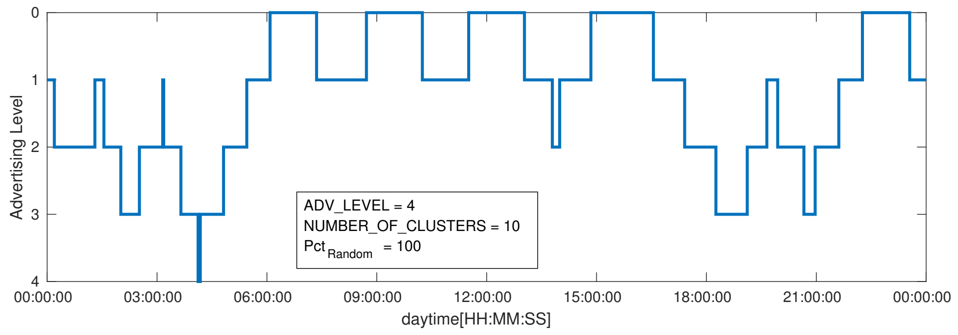
4.1.1 and Section 4.3.3, respectively. As a matter of lucidity, both measurements are summarized in Table 5.3, where DS is the time spent in deep sleep. We assume an Android smart phone as scanner, performing continuous scanning (scan interval = scan window = 5000 ms).

Based on this results, we can estimate the consumed energy while using static advertising, but also while AA is used. Thanks to the daily schedule (see Figure 5.12) we know how much time is spent in each level and thus, how much time is spent using a certain advertising interval. Table 5.4 illustrates the measured mean power consumption over a period t of static advertising with an interval of 417.5 ms ($P_{S,meas.}$) and Adaptive Advertising ($P_{AA,meas.}$), dependent on the three different simulation scenarios (random, single household, and family household). The deviation (Dev.) of the estimated power consumption ($P_{AA,est.}$) and $P_{AA,meas.}$, and the gain of AA (Gain), meaning the reduction of the mean power consumption, are expressed as percentage value. The results allow three observations:

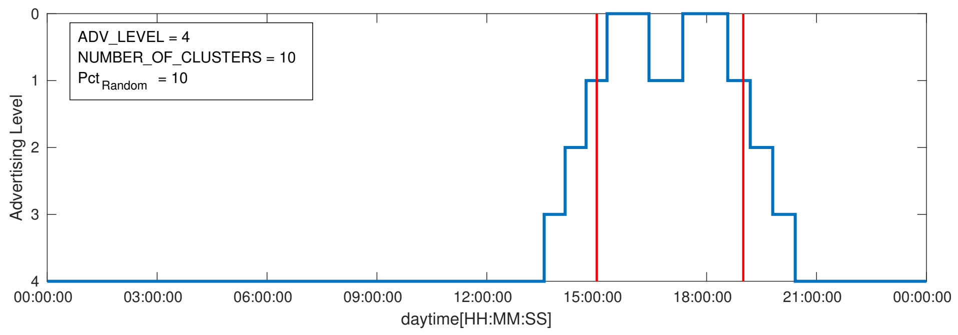
1. The estimation of the mean power consumption with the model is accurate. It allows us to predict the energy consumption within an **accuracy of ± 2%**.

Scenario	$P_{S,meas.}$[mW]	$P_{AA,est.}$[mW]	$P_{AA,meas.}$[mW]	Dev.[%]	Gain[%]
Random	0.4689	0.5783	0.5895	-1.94	+23.33
Single	0.4689	0.2456	0.2414	+1.68	-48.52
Family	0.4689	0.3629	0.3681	-1.43	-21.50

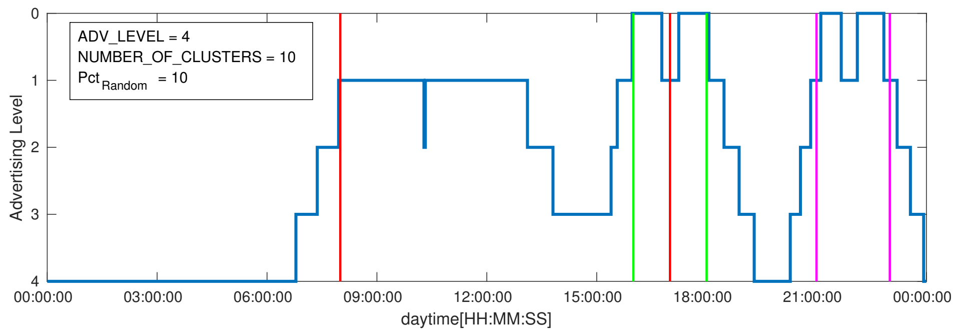
Table 5.4: Estimated ($P_{AA,est.}$) and measured ($P_{AA,meas.}$) mean power consumption of AA in comparison with the mean power consumption of static advertising ($P_{S,meas.}$).



(a) Schedule for a random time value distribution using five advertising levels.



(b) Schedule for the single household scenario using five advertising levels.



(c) Schedule for the family household scenario using five advertising levels.

Figure 5.12: Resulting schedules based on different data: random (a), single household (b), and family household (c).

2. AA reduces the mean power consumption in most of the cases significantly. The exact value depends on the value distributions. For our scenarios, we are able to **reduce the mean power consumption** by some more than **48%**. The more a user shows deterministic, routine behavior (most of the values are around certain time points), the lower the power consumption of AA is.

3. AA **increases the mean power consumption** in cases where the data values are **spread randomly** over the whole 24 hours. Therefore, clusters are also spread over the full range resulting in more time spend on advertising level 0 and thus, an increase in power consumption of **23%** is observable.

Scenario	$\text{Latency}_{\text{S}}$[ms]	$\text{Latency}_{\text{AA}}$[ms]	Better[%]	Equal[%]	Worse[%]
Random	408.3	172.4 - 4145.8	35.7	28.7	35.6
Single	408.3	172.4 - 4145.8	57.0	34.3	8.7
Family	408.3	172.4 - 4145.8	51.0	34.7	14.3

Table 5.5: Influence of AA on the mean device discovery latency ($\text{Latency}_{\text{AA}}$) in comparison with static advertising ($\text{Latency}_{\text{S}}$) for different scenarios (random, single household, and family household).

We showed that AA is possible to change the amount of consumed energy, but how does this influence the user experienced latency? Changing the advertising interval has a direct impact on the device discovery latency (see Table 5.3). Therefore, we evaluate for each scenario (random, single household, and family household) for which amount of interactions the device discovery latency improves, stays the same, or degrades. By the knowledge of the schedules (see Figure 5.12) and the used advertising level mapping (see Table 5.2) we know which advertising interval is used at which point in time. Table 5.3 gives us the mean device discovery latency for each interval. Based on the created time values and the resulting schedule, we know how many time values occur during which advertising level. The resulting device discovery latency ($\text{Latency}_{\text{AA}}$) is compared with static advertising ($\text{Latency}_{\text{S}}$, assuming an advertising interval of 417.5 ms) and is illustrated in Table 5.5. AA regulates the mean device discovery latency between a certain interval. The exact values depend on the mapping of advertising intervals to advertising levels (see Table 5.2). For our example, the mean device discovery latency is between **172.4 ms** and **4145.8 ms**. We distinguish between three different cases for all three scenarios in comparison with static advertising:

1. Better: an improvement of the mean device discovery latency is achieved for the stated percentage of all time values, which means, that advertising level 0 is used. In our case, this reduces the mean device discovery latency to 172.4 ms, which reduces the time by more than **57%** compared to static advertising ($T_{ADVI} = 417.5$ ms).

2. Equal: neither an improvement nor a deterioration of the mean device discovery latency is achieved for the stated percentage of all time values. This means, that the time is spend on advertising level 1, which is the same as static advertising is using.

3. Worse: for the stated percentage of all time values a deterioration of the mean device discovery latency is introduced, meaning advertising level 2 or higher is used. Dependent on the current level this increases the mean device discovery latency up to 4145.8 ms.

As illustrated in Table 5.5, the benefit of using AA is different for each scenario (random, single household, or family household). The more deterministic the usage behavior of a user is, the higher is the benefit of AA. This argument is confirmed once we combine *Better* and *Equal* values of each scenario: once a random time value distribution was chosen, AA performed better or equal in 64.4% of all cases. In contrast, using a highly deterministic behavior (single household) with a tenth of randomness, AA performed better or equal in 91.4% of all cases. Having a reduced deterministic behavior (family household) decreases this value to 85.7%.

Limitations. AA increases the mean device discovery latency during periods of low device activity. The schedule is calculated on a daily basis. This introduces an issue, once user behavior is dramatically changing from one day to the next day. In such cases, users experience high latency and energy is wasted in periods where it would not be needed. Furthermore, as targeted by our simulation scenarios, there is a part of time values that are not part of a predictable behavior, so called random time values. They can occur each day with a different percentage. Once those random time values occur in periods where a high advertising level is used, the user experiences a high latency.

5.4.4 Outcome

The three different used time value types (random, single household, and family household) prove the characteristics of the algorithm Adaptive Advertising - User Behavior (AA-UB): we developed an algorithm that can react adequate to different data input. Parameters that can be set developer-sided at compile time make the algorithm highly versatile and available for different application areas: dependent on the use case, the algorithm can be fine-tuned in order to achieve the best results regarding power consumption and device discovery latency.

We examined that the benefit of AA depends on the determinism and regularity of user behavior and even with a decent amount of randomness (10%) a significant **reduction of energy consumption and device discovery latency** is achieved: for both, simultaneously, more than **50%** are possible.

We showed a mathematical approach wherewith a **prediction of the mean power consumption** with a **deviation of under 2%** is possible.

In order to prevent the limitations mentioned in Section 5.4.3, the concept of a Range Extender is introduced in the next chapter.

Chapter 6

Range Extender

In this chapter, the concept of Range Extender (RE) is introduced. In Section 6.1 we describe the principle of Range Extender. The aim of RE is to extend the Adaptive Advertising (AA) algorithm, presented in Chapter 5. RE allows to achieve a satisfying usability even when user behavior is unpredictable or suddenly changing. It does so by notifying BLE devices about the presence of other nearby BLE devices. Notified devices may adapt their advertising interval in order to reduce the device discovery time of potential connection partners. Two different modes, Permanent Connection (PC) and Temporary Connection (TC), are described. Although the description of the concept is explained on the basis of the NSL, it can be applied on any use case where three or more BLE devices are interacting with each other.

The design, the implementation, and the evaluation are based on the specific use case of the Nuki Smart Lock, which is described in Section 6.2.

In Section 6.3, three different design possibilities of the RE are introduced: *Powerful NSL*, *Powerful RE*, and *Current System and RE*. They differ in application- and smart lock-sided changes that have to be performed in order to use the RE functionality.

Section 6.4 describes the implementation of *Powerful NSL* in both PC and TC mode.

In Section 6.5, we evaluate *Powerful NSL* in both modes according to energy consumption and device discovery latency. The evaluation consists of two different scenarios, *high traffic* and *standby*.

6.1 Concept

As described in Section 4.2.1, the range of BLE is very limited. Furthermore, obstacles can be between the NSL and the smart phone. One of those obstacles is the door on which the NSL is mounted. Depending on the material of the door, the range may decrease significantly (e.g., a metal door). The usability depends heavily on the range at which the NSL and the smart phone can interact the first time. The device discovery, connection establishment, exchange of unlocking information (authentication), and lock/unlock mechanism take time (see Figure 6.1). To avoid that the user is waiting in front of the locked door, this time should be shorter than the time from the first interaction between the two devices till the user has reached the door. The time spent on device discovery is highly *variable* and depends on several factors, that are described in Chapter 4. In con-

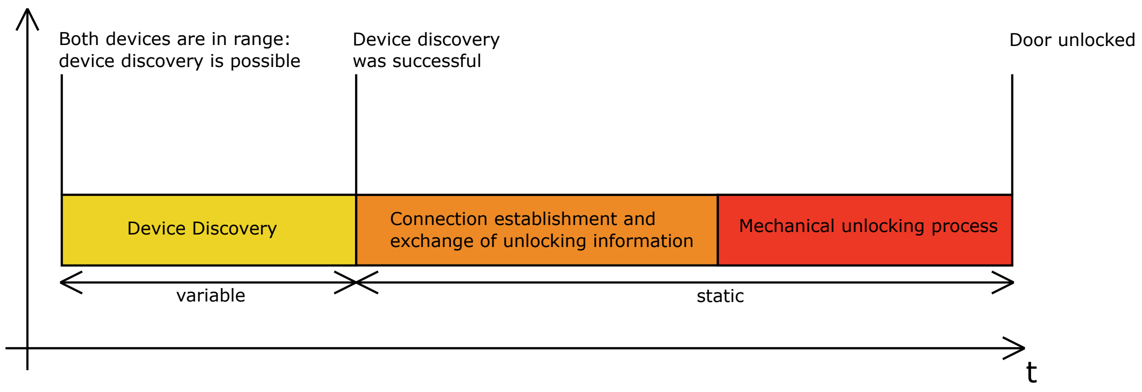

Both devices are in range:
device discovery is possible
Device discovery
was successful
Door unlocked
Device Discovery
Connection establishment and
exchange of unlocking information
Mechanical unlocking process
variable
static
t

Figure 6.1: Phases of the unlocking mechanism of the NSL: starting with the device discovery between smart phone and NSL, followed by a connection establishment and exchange of unlocking information (authentication), and concluded by the mechanical unlocking process (not to scale).

trast to that, the connection establishment, the exchange of unlocking information, and the mechanical unlocking process are not adjustable, as they are realized by Nuki. Thus, these time periods are named *static*.

In Section 4.2.2, we calculated, based on the maximum LOS distance of 14 m, a maximum available time for device discovery of 10.08 seconds. The specific use case NSL introduced two additional, static time consuming phases (connection establishment, mechanical unlocking), illustrated in Figure 6.1. Furthermore, as mentioned above, the maximum communication distance of 14 m may decrease by environmental reasons. These new circumstances decrease the maximum time available for device discovery and introduce the need of a range extender.

Increasing the range shifts the point where both devices are able to interact the first time to an earlier point in time. This increases the time available to complete the fore-mentioned steps.

To avoid that a user has to wait in front of the locked door, a range extender could be used. This **range extender** (RE) could be another BLE device placed outdoors acting as an additional hop between the smart phone and the NSL. This device could be used to extend the range of the smart lock.

The task of the RE is to detect nearby, approaching BLE devices. After a new BLE device (e.g., smart phone) was discovered by the RE, the RE notifies the NSL about the smart phone. In case multiple smart locks are present, the RE is able to notify all of them. Furthermore, it is possible that a single smart lock receives information about nearby devices from multiple REs, e.g., a smart lock itself, supporting the RE feature, could notify other smart locks. After the notification was received, the NSL may switch to a faster advertising interval in order to decrease the device discovery latency. Once the smart phone gets in range of the NSL, thanks to the faster advertising interval used by the NSL, the phone is able to detect the lock faster. It can send a connection request and in a further step unlock the door earlier. This reduces the user experienced latency for the difference in device discovery latency dependent on the selected advertising interval.

Challenging is the fact that approaching devices perform scanning to discover the advertising packets of the NSL. Therefore, the RE can not perform scanning in order to get knowledge about the presence of scanning devices in its close neighborhood. Hence, we need to turn around the device discovery process of BLE: the RE itself advertises periodically its presence containing information about a range extender service. For details about advertising and its data structure see Section 2.1.2.1 and Section 2.2.1. This allows a scanning device, e.g., a smart phone, to discover the RE. Once the smart phone received an advertising packet, it sends by default a scan request. If this is not the case, the running Nuki app has to initialize the scan request. This happens after the smart phone analyzed the packet payload. If the packet contains the UUID (see Section 2.1.6.1) of the range extender service, the smart phone knows that the RE may be of relevance and sends a scan request. By sending the scan request, the RE gets notified about the presence of a smart phone that may be relevant for the NSL. If the smart phone is actually relevant or not depends on the relationship of trust between smart phone and NSL. If they were paired previously, the smart phone has the privileges to unlock/lock the door. Else the phone is not a trusted device and can not perform any actions. The decision about the relevance of a phone for a certain NSL can not be made by the RE!

The information exchange between RE and smart lock is connection based. Therefore, the RE has to perform scanning to discover the advertising smart lock and to initiate the connection with the smart lock. We distinguish between two modes in which the RE could operate, **Permanent (PC)** or **Temporary (TC)** Connection mode.

6.1.1 Range Extender - Permanent Connection (RE-PC)

In this scenario, a connection between RE and NSL is established after the first notification is sent and maintained permanently. All further notifications are sent using the already established connection. While holding a connection, both devices need to advertise: the RE needs to detect possible nearby devices and the NSL needs to stay responsive for possible connection requests by trusted smart phones. This requires that a BLE device can sustain multiple connections simultaneously.

6.1.2 Range Extender - Temporary Connection (RE-TC)

Here, the range extender connects to the NSL only when needed: in this case a connection between the extender and the NSL is only created when a nearby device was detected by the RE. The RE initiates the connection, exchanges information about nearby devices, and terminates the connection afterwards. In this mode, again both devices, NSL and extender, need to advertise. The RE advertises in order to be able to be detected by approaching smart phones. The NSL needs to advertise in order to be discoverable for a connection establishment with the RE or with a smart phone.

In contrast to RE-PC, this mode is applicable on devices that support only one active BLE connection at any point in time. The whole time where both devices are connected and thus not discoverable for other BLE devices is very short.

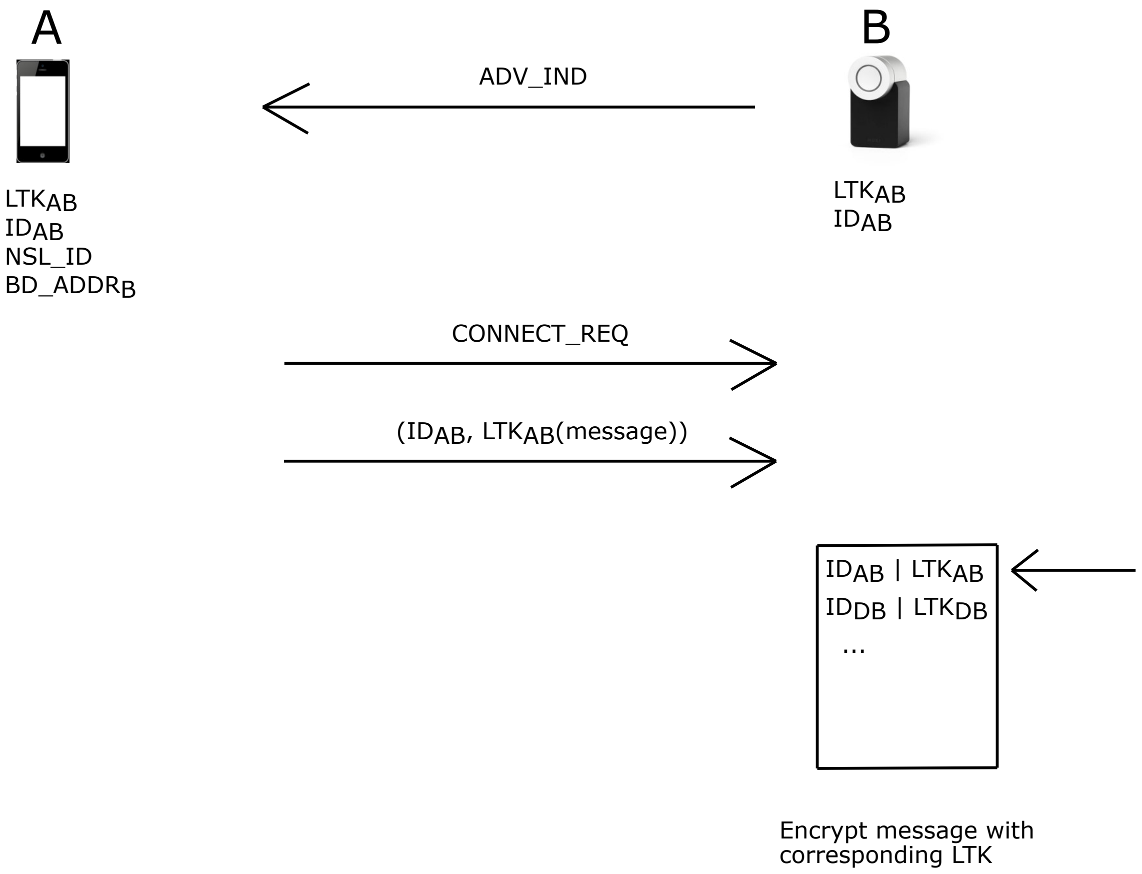


Figure 6.2: Sending messages from a smart phone to a paired Nuki Smart Lock (NSL).

6.2 Use Case: NSL

The special design of the BLE communication between smart phone and Nuki Smart Lock (NSL) (see Section 2.2.1) is the reason for several challenges for the design of a Range Extender (RE). The current system is illustrated in Figure 6.2. After the phone (device A) is paired with the NSL (device B), both devices derive the shared Long Term Key (LTK_{AB}) with a Diffie-Hellman key exchange. The smart phone and the NSL exchange the device and lock specific authentication ID. Furthermore, the smart phone knows the NSL_ID, an identifier that is unique for each Nuki Smart Lock, and the real BLE device address of the NSL BD_ADDR_A. The NSL does not use the privacy feature of BLE and therefore its address does not change during lifetime (see Section 2.1.2.8).

Once the smart phone is in range of the NSL and is able to receive the advertising packets of the NSL, it can start to send a connection request (CONNECT_REQ) to the NSL. After the connection is established communication can take place on the BLE data channels. Each request or action performed by the phone is now encrypted with the shared LTK with exception of the ID. The NSL uses this unencrypted ID in order to find the corresponding LTK. With LTK_{AB} the message can be decrypted and interpreted.

The aim of the Range Extender (RE) is to inform the NSL on the possible approach of a smart phone. To avoid device tracking, smart phones usually use resolvable private addresses. Private means that during communication the device uses an address (addr_randres) that is not the BD_ADDR and changes over time, e.g., every 15 minutes. Resolvable means that with the possession of the device specific Identity Resolution Key (IRK) it is possible to recover BD_ADDR from the addr_randres. More information about address types can be found in Section 2.1.2.8. The IRK can be exchanged between devices after a successful pairing process. This enables tracking of a device for all BLE devices that know its IRK!

6.3 Design Possibilities

We distinguish between three possibilities that allow to inform the NSL about the approach of a BLE device. They differ in app- and NSL-sided changes that have to be made in order to make use of the RE functionality. To start with, the concept of the **Powerful NSL** is introduced. There, the evaluation of the relevance of nearby devices is kept in the NSL. In contrast to that, **Powerful RE** decides itself if nearby devices are of relevance for a specific smart lock and informs the smart lock only if relevant information is available. Finally, **Current System and Range Extender** tries to introduce as little changes as possible according to the existing implementation of the NSL.

6.3.1 Powerful NSL

The evaluation of the relevance of nearby devices is kept in the NSL (see Figure 6.3). This demands that during the pairing process of smart phone and NSL the smart phone shares its IRK and its BD_ADDR with the NSL. This enables address resolution in a later point in time. The NSL needs to store both values together with the already existing values of ID and LTK.

In order to make use of the RE functionality, the NSL subscribes to a custom service that enables notifications about nearby devices.

The RE is advertising scannable advertising packets. The smart phone has to perform active scanning. Android devices perform active scanning per default, while phones running with iOS perform passive scanning. For iOS devices active scanning has to be enabled over the Nuki application. Once the smart phone receives a scannable advertising packet (ADV_SCAN_IND) it sends a scan request (SCAN_REQ) back to the RE. Thanks to this request, the RE knows the resolvable private address of device A ($\text{addr\_randres}_{\text{A}}$). The RE does not have knowledge about the IRK_{A} and is not able to find out $\text{BD\_ADDR}_{\text{A}}$.

The RE holds a list of addresses from devices who sent a scan request. On each appearance of a new address (remember: it can be the same device that changed the address) it notifies the NSL about the new address. The NSL has to perform address resolution with the stored IRK. If the private resolvable address and the IRK are of the same device, it results in a valid BD_ADDR. The validity of this operation can be checked as the NSL has stored IRK and matching BD_ADDR. If the NSL has stored more than one IRK, it has to repeat address resolution for each IRK, until either address resolution is successful or all IRKs have been evaluated.

If address resolution was successful, the NSL knows that a device that has permissions to perform locking actions is close and further steps can be initiated. When address

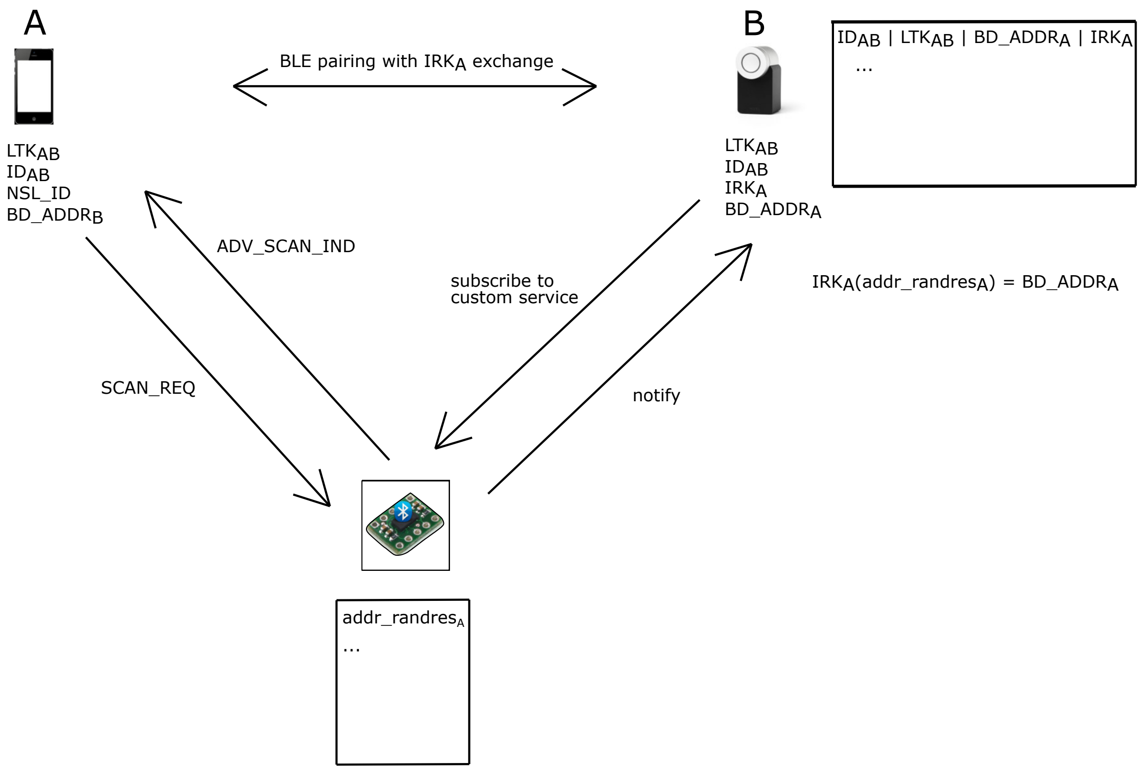

A
B
BLE pairing with IRKA exchange
IDAB | LTKAB | BD_ADDRA | IRKA
...
LTKAB
IDAB
NSL_ID
BD_ADDRB
LTKAB
IDAB
IRKA
BD_ADDRA
ADV_SCAN_IND
subscribe to custom service
IRKA(addr_randresA) = BD_ADDRA
SCAN_REQ
notify
addr_randresA
...

Figure 6.3: Powerful NSL: range extender informing the NSL about approaching devices and their addresses. The NSL performs address resolution in order to uniquely identifying the approaching device and decides if it is relevant.

resolution was unsuccessful, no relevant devices are currently in the neighborhood and therefore no further actions have to be performed.

- Advantages:

  1. Centrality: the decision about the relevance of a detected device is kept in the NSL. This allows to add an arbitrary number of REs.
  2. Simple RE: the RE looks for nearby devices and lets the NSLs decide if they are relevant or not.
  3. Selectivity: the NSL can decide which REs are of interest and does not have to subscribe to all of them. The NSL can also temporary subscribe to the service of a RE and unsubscribe at a later point in time.
  4. Privacy: the devices IRKs are only shared with the NSL and do not leave the NSL. REs are not able to track devices.
  5. Diversity: any device supporting the custom service can act as RE. For example, other NSLs can act as range extender as no device has to know the IRKs of other devices.

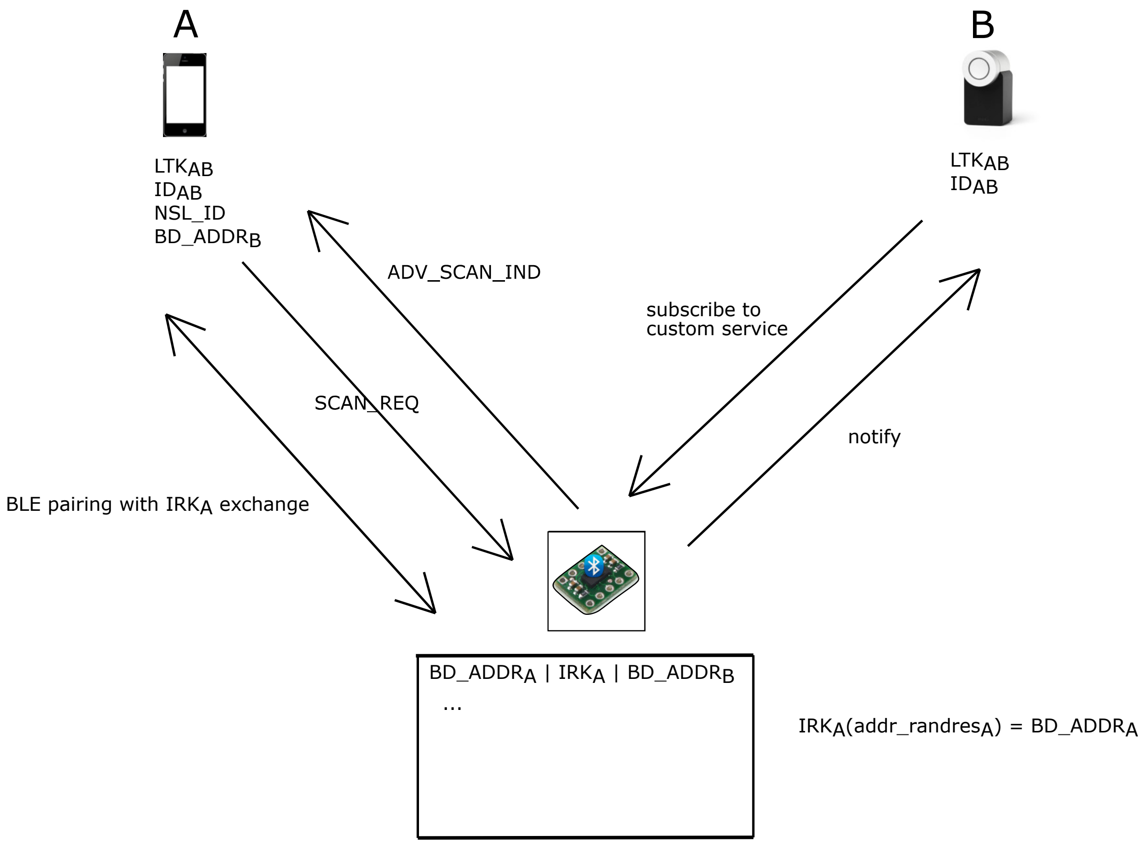

A
B
LTK$_{AB}$
ID$_{AB}$
NSL_ID
BD_ADDR$_B$
LTK$_{AB}$
ID$_{AB}$
ADV_SCAN_IND
subscribe to custom service
SCAN_REQ
notify
BLE pairing with IRK$_A$ exchange
BD_ADDR$_A$ | IRK$_A$ | BD_ADDR$_B$
...
IRK$_A$(addr_randres$_A$) = BD_ADDR$_A$

Figure 6.4: Powerful RE: range extender identifies approaching devices, performs address resolution and informs the NSL only if the approaching device is of relevance for a certain NSL.

- Disadvantages:
    1. Complex NSL: the functionality on the NSL has to be extended. IRKs and BD_ADDRs of all devices with locking permissions have to be exchanged and stored. Furthermore, on each notification about a new device, address resolution has to be performed.
    2. Active Scanning: the phone has to perform active scanning in order to inform the RE about its presence. Therefore, the Nuki application has to be adapted.

6.3.2 Powerful RE

As shown in Figure 6.4, the evaluation of the relevance of nearby devices is outsourced to the RE. This means that the RE decides which devices are relevant for a specific NSL and inform the NSL about necessary actions. This implies that the RE is in possession of the IRKs of phones that have access to the NSL. Therefore, users that want to use the RE have to pair their phone once with the RE. During the pairing process, the phone shares its IRK and its BT_ADDR with the RE. Furthermore, the phone forwards the BD_ADDR of the NSL it has access to. The RE stores all three values and creates a new entry for each NSL if a phone has access to multiple locks.

As for the scenario above, the NSL subscribes to a custom notification service in order to get notified about approaching devices. The RE is advertising scannable advertising events (ADV_SCAN_IND). Again, the phone has to perform active scanning in order to inform the RE about a scan request (SCAN_REQ) and therefore about its presence. The private resolvable address of the phone (addr_randres_A) can be read from the scan request. Based on this address, the RE performs address resolution. If address resolution is successful, all NSLs (to which the approaching smart phone has access to) get notified and the NSL can initiate further actions.

- Advantages:
  1. Simple NSL: as all additional features are placed on the RE, no changes have to be performed on the NSL.
  2. User decision: the user decides which REs are used and has full control over the network. REs and NSLs of different users are separated from each other.
- Disadvantages:
  1. User interaction: each used RE has to be paired once to the smart phone of the user.
  2. Complex RE: each RE has to evaluate which devices are relevant for subscribed NSLs and notify the corresponding locks.
  3. Limited network: the user can only make use of its own REs and NSLs. Possible nearby REs or other locks that are not under control of the user can not support to the functionality.
  4. Active Scanning: the phone has to perform active scanning in order to inform the RE about its presence. Therefore, the Nuki application has to be adapted.

6.3.3 Current System and RE

With this approach a solution with as little changes as possible should be found in order to use a Range Extender together with the current system. As we know about the two previous possibilities, smart phone sided active scanning is necessary in order that the RE can detect the smart phone. Therefore, active scanning has to be enabled over the Nuki application. As explained before, the RE notifies the NSL about nearby devices. In contrast to the illustration in Figure 6.3 we do not exchange phones' IRKs. As a result, the NSL is not able to resolve the resolvable private addresses of the devices. Therefore, it would be unnecessary to send the resolvable private addresses from the RE to the NSL. We could substitute this with a short notification stating that one or more BLE devices are in the neighborhood. Of course we are not able to make a decision about the relevance of such an notification.

- Advantages:
  1. Few changes: the functionality of the RE can be seen as simple extension to the current system. No changes have to be performed on the NSL. All we need

is enabled active scanning which can be achieved with an update to the Nuki smart phone app.

- Disadvantages:
    1. Relevance of devices: no statement about the relevance of a received notification can be made because BLE device addresses can not be resolved.
    2. Static devices: devices that are permanently active scanning in the close neighborhood of the RE can lead to continuous notifications if they change their addresses frequently. The RE thinks a new device appeared every time the address is changed. Sending notifications continuously drains the battery of both devices (RE and smart lock).
    3. Active Scanning: the phone has to perform active scanning in order to inform the RE about its presence. Therefore, the Nuki application has to be adapted.

6.4 Implementation

In this section, we describe the implementation of the **RE - Powerful NSL**, introduced in Section 6.3.1. This approach keeps decisions about further actions in the NSL (e.g., no sensitive information leaves the lock). The whole network is easily expandable by adding new REs or other BLE devices that support the RE functionality. Furthermore, the user does not have to perform any setup in order to use the new functionality. This keeps the system simple but enables to ignore devices that are not relevant for a specific smart lock.

For the smart lock, again, the CDK (see Section 2.2.1) is used. Its BLE chip comes up with a problem: the chip can only hold one connection at any point in time. This fact blocks any further interactions with other devices. During a connection no advertising or scanning is possible. This disables the chip for all further actions once a connection is established. For our case and the PC mode this means, that once the RE would detect a smart phone which probably wants to connect to the smart lock, the RE would need to drop the connection with the NSL. The NSL itself would not be able to advertise itself, making a phone sided discovery impossible. Still both modes are implemented and evaluated, as a substitution of the chip is planned by the company soon.

The nRF52 BLE chip was chosen as range extender. This decision was made in order to be open to further development with respect to the new BLE v5.0 standard (see Section 2.1.9). The nRF52 is available as Development Kit (nDK). More information about the hardware can be found in Section 2.2.3.

The implementation of the RE can be subdivided into five main parts, that are discussed in the following sections:

1. Advertising: advertising packet structure and advertising policy in order to be discoverable for other BLE devices.

2. Scanning and connection establishment: used scanning policy for the sake of detecting smart locks, followed by a connection establishment.

3. Scan requests: scan request notifications and their benefits.

4. Nearby device service: service provided by the RE in order to inform NSLs about nearby devices.

5. Modes: differences of the two main supported modes, RE-PC and RE-TC. Additionally, there is the possibility to enable event driven scanning that subdivide the PC and TC mode further.

6.4.1 Advertising

The range extender is using static advertising. The used advertising interval can be selected at compile time via the parameter *ADVINT*, depending on device discovery latency requirements. Note that the device discovery latency depends not only on the advertising interval of the RE but also on the used scan interval and scan window of the smart phone (see Section 4.3.3).

6.4.2 Scanning and Connection Establishment

The scan interval can be set with the define statement *SCANINT* in *config.h*. According to the results of the device discovery latency, presented in Section 4.3.3, we scan with the OSP scanning policy (see Section 4.3) according to the slowest possible advertising interval of our counterpart (4000 ms).

The RE scans for advertising packets with Nuki's key turner service (see 2.2.1) in order to identify smart locks. Once a smart lock was found, the RE initiates a connection by sending a connection request to the NSL. For the initial connection establishment, Accelerated Connection Establishment (ACE) is used [Mik14]. This approach selects a shorter connection interval for the connection establishment in order to be able to faster terminate connections that fail to establish. Furthermore, a fast connection interval allows the NSL to discover attributes provided by the RE faster (see Section 2.1.6.1). According to ACE, after a successful connection establishment, the connection interval is adapted by a connection parameter update (see Section 2.1.2.6).

After a successful service discovery, the NSL knows attribute handles used by the RE and, as a consequence, is able to subscribe to the nearby device service provided by the RE. Of course it is possible to revoke the subscription at any point in time. For more information about how service discovery and notifications work see Section 2.1.6.1.

Once subscription to the nearby device service was successful, the RE sends the current entries of the nearby device list to the smart lock. As the connection is established, attribute handles are known, and data is received, we want to conserve energy. Therefore, the NSL sends a connection parameter update request that includes slower connection intervals and also a higher timeout. Still, we do not want to allow the slave to skip connection events. This would increase the latency unnecessarily and thus a slave latency of 0 should be selected. Further proceedings are dependent on the selected mode. Differences can be found in Section 6.4.5.

6.4.3 Scan Requests

Scan requests are necessary in order that a BLE advertiser (e.g, the RE) is able to get knowledge about the presence of a BLE scanner (e.g., a smart phone). Therefore, the scanning device has to perform active scanning, meaning that once the scanning device received an advertising packet it sends a scan request to the advertising device. By the reception of the scan request the advertiser gets informed about the presence of the scanning device.

In Zephyr OS the scan request feature can be enabled by setting the configuration variable *CONFIG_BT_CTLR_SCAN_REQ_NOTIFY* in the project's configuration file accordingly. Although this feature is defined in the BLE specification [Blu14], it is still under development in Zephyr, and therefore, even when setting the previously mentioned configuration variable, scan requests are disabled. In order to make use of the feature, we have to modify the kernel slightly.

In the file *ctrl.c* which is located in *zephyr/subsys/bluetooth/controller/ll_sw* one can find several if defined statements dependent on the configuration variable *CONFIG_BT_CTLR_SCAN_REQ_NOTIFY*. Taking a closer look at the function *isr_rx_adv()* shows that the generation of the scan request event by *isr_rx_adv_sr_report()* is suppressed by a if(0) statement. Replacing the zero value by one enables the event generation.

Unfortunately, so far, the triggered event can not be handled on application level. All we notice is a debug message stating that a scan request was received. Thus, again, we have to slightly change the kernel.

In function *encode_control()* in file *hci.c*, which can be found in *zephyr/subsys/bluetooth/controller*, one can observe another define statement dependent on the above mentioned configuration variable. On a received scan request the function *le_scan_req_received()* is called. We substitute this function by our own function *ndAddDevice()* that is part of the Nearby Device Service and thus more information can be found in Section 6.4.4

6.4.4 Nearby Device Service (NDS)

The nearby device service (NDS) is a service that informs interested devices about recently detected devices. In our use case, the RE implements this service in order to inform the smart locks about approaching devices. In this section, a general service description is given. Furthermore, it is explained how the RE maintains the list of nearby devices.

A UUID was assigned to the service (see Section 2.1.6.1). This 128-bit identifier is created randomly and uniquely identifies the service. Devices offering this service shall maintain a list with addresses of devices that where seen recently. Furthermore, connected devices shall be able to enable notifications or simply read the list of devices. If notifications are enabled, the device should provide the content of the device list corresponding to the following description. Each notification starts with a byte indicating how many addresses are contained in a single notification. For each address, transmitted in little-endian order, 7 bytes were send, where 1 byte is used for the type of the address (private/public) and 6 bytes are used for the address itself. One single notification contains at most 2 addresses in order to be compatible with a minimal MTU size of 23 bytes, resulting in an actual payload of 20 bytes (see Section 2.1.6.1): one byte is used for opcode, wherewith the packet can be marked as notification. Two additional bytes are needed to identify the specific value, the actual device list. Therefore, the server specific attribute handle is used. Hence, in total 8 (1 address) or 15 (2 addresses) bytes are transmitted for each notification. If not all devices fit into one single notification, additional notifications with the same structure are sent. Remember, more than one notification can be sent in one

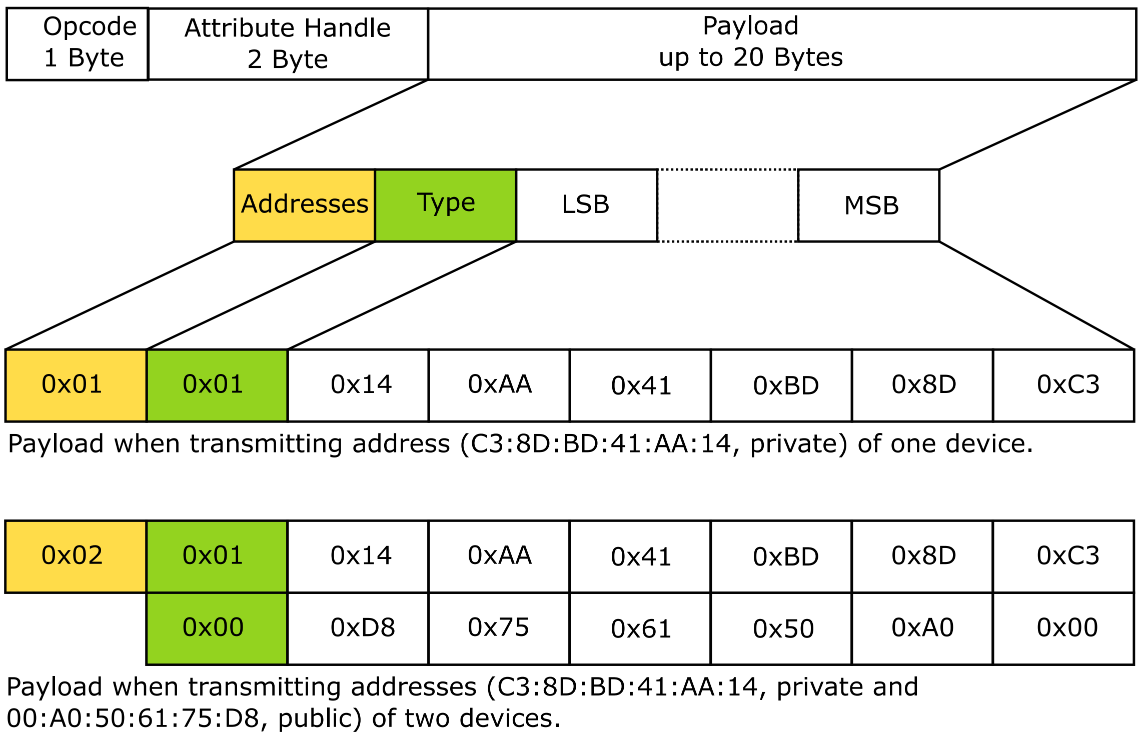

Opcode
1 Byte
Attribute Handle
2 Byte
Payload
up to 20 Bytes
Addresses
Type
LSB
MSB
0x01
0x01
0x14
0xAA
0x41
0xBD
0x8D
0xC3
Payload when transmitting address (C3:8D:BD:41:AA:14, private) of one device.
0x02
0x01
0x14
0xAA
0x41
0xBD
0x8D
0xC3
0x00
0xD8
0x75
0x61
0x50
0xA0
0x00
Payload when transmitting addresses (C3:8D:BD:41:AA:14, private and 00:A0:50:61:75:D8, public) of two devices.

Figure 6.5: Packet structure of a single notification when transmitting one or two different device addresses with an MTU of 23 bytes.

connection event (see Section 2.1.6.1).

Applying the mentioned rules the PDU of a single notification looks like illustrated in Figure 6.5.

Each range extender has a list containing detected devices. The list is a structure containing the following fields:

- number_of_devices: integer to keep track of the amount of devices that are part of the list.
- devices[]: array with a maximum of MAX_ND_LIST_ENTRIES entries (configurable in *config.h*). In this array, all addresses of discovered devices get stored. Each address consists of its type (1 byte) and its actual value (6 bytes).
- timestamp [MAX_ND_LIST_ENTRIES]: in this array, a time stamp for each discovered device is hold. The time stamp is based on the up time of the system. Once a device reappears, the time stamp gets updated accordingly.

On each received scan request (see Section 6.4.3) the function *ndAddDevice()* gets called. The scan request contains the address of the requesting device. This address is used to check if the device was already seen recently. Upon a scan request, we distinguish between two cases:

- Address already part of the list: in the first case, the corresponding time stamp gets updated to the current up time of the system. No notifications are sent.

Main Mode	Sub Mode
PC_TC_MODE	EDS
PC	disabled
PC	**enabled**
TC	disabled
TC	**enabled**

Table 6.1: Supported modes of the RE. Recommended modes are bold.

- New address found: in the second case, if advertising is active we disable advertising and start scanning (if event driven scanning was selected). This allows us to detect smart locks. The discovered device is added to the list with the current time stamp. Afterwards, if not already established, connections are created and subscribed devices get notified at the next connection event.

Independent from a possible previously detected presence of a device, at each received scan request, the devices are updated regarding timeliness. If more time than *LAST_SEEN_THRESHOLD_MS* passed since the last received scan request of a device, then the device is removed. The threshold can be adjusted in *config.h*. This removal of devices keeps the list small and avoids that smart locks adjust their advertising interval for devices that are not relevant anymore as too much time passed.

6.4.5 Modes

As mentioned previously, the RE can be used in two different modes: Permanent Connection (PC) and Temporary Connection (TC) mode. Most of the implementation remains the same, but still there are some differences that are explained in Section 6.4.5.1.

Furthermore, by the usage of **event driven scanning (EDS)** the two main modes can be subdivided. Its description and influence is discussed in Section 6.4.5.2.

An overview of all modes can be found in Table 6.1. In order to achieve the best results regarding latency and energy consumption, we suggest to use the modes marked bold. More details can be found in Section 6.5.

6.4.5.1 PC / TC

The selection between the two main modes, PC and TC, is performed via the configuration variable *PC_TC_MODE* and can be found in *config.h*. Setting it to 0 enables PC mode, 1 executes TC mode.

- PC: connections are established whenever possible. This means the RE sends a connection request once a smart lock was detected. Once connected, the connection is maintained permanently, meaning that the RE itself does not disconnect. Reasons for disconnect are link loss or if the connection is terminated by the remote device.

- TC: in TC mode, a connection is established only when needed and disconnected when not needed anymore. The connection is hold temporary. Similar to PC, this assumes that the RE was scanning in order to be able to detect smart locks. Once the subscription for notification is received, the RE sends the current content of the device list. It can take up to connection interval seconds till the notification is actually transmitted. Therefore, the connected device is not disconnected directly after the notification. Instead, the device is marked as disconnectable and is disconnected during the next main loop which is sleeping for *LOOP_DURATION*. Also this parameter can be changed in *config.h*. Because we are using ACE [Mik14], the initial connection interval is relatively short. The update takes place after the first received notification. Note, that *LOOP_DURATION* shall be bigger than four second, which is the longest possible connection interval. This ensures that a second connection interval is granted to both devices. This could be necessary if not all data fits in one connection interval (see Section 2.1.6.1). Afterwards, the connection is terminated by the RE.

6.4.5.2 Event Driven Scanning (EDS)

Event driven scanning (EDS) can be enabled by setting *EDS* in *config.h* to 1.

Continuous scanning is very expensive as the radio has to be turned on over the whole period. Furthermore, it may cause troubles with other tasks accessing the radio, e.g., advertising or maintaining a connection. The radio can be seen as resource which has to be shared between different tasks. In our case we would need to scan, advertise, sending data and maintain a connection (only PC mode). We can brake that down to the three main events of BLE: advertising, scanning and connection event (see Section 2.1.2.1, Section 2.1.2.2, and Section 2.1.2.5). Fortunately, Zephyr OS manages the radio resource autonomously, but we have to consider the radio constraints. Zephyr prioritizes concurrent tasks. The highest priority is assigned to connection events, followed by advertising events and scanning events. This leads to the case that every scan window can be interrupted by an advertising event or a connection event. After the preemption of an event, the event is not resumed afterwards and will only be revived on the next corresponding interval.

Thus, the selection of the used advertising interval, connection intervals, and the schedule of connection events (e.g., when multiple devices are connected to the RE) limit the duration of scan windows. This fact has to be considered during the implementation and the energy measurement.

Therefore, we introduced an event driven scanning policy. By using EDS, the RE is scanning only when it is needed. This means, the RE starts scanning once it has to send out notifications about nearby devices. The duration of the scanning period is defined in *EDS_DURATION* and should be at least three times the scan interval. This enables scanning on all three advertising channels.

To avoid interruptions of the scan window, caused by advertising events, we **disable advertising** while scanning. On the one hand, this allows us to use scan windows according to OSP in order to not cause unpredictable device discover latency. On the other hand, disabling advertising while scanning leads to the fact that after a successful device discovery the RE is not detectable immediately for new devices that might appear during the scan period. In a further step this leads to a later received scan request and thus nearby devices may not be detected in a reasonable time. With our proposal using three times OSP scanning policy this time can take up to 12.1 seconds.

6.5 Evaluation

We introduced two main modes, permanent connection (PC) and temporary connection (TC) with further possibility of enabling the feature event driven scanning (EDS).

In Section 6.5.1 we describe the measurement setup which is used for the evaluation of the system.

Depending on the use case, one mode may be better than another one. In Section 6.5.2 and Section 6.5.3 we examine the energy consumption and the latency of the modes PC and TC respectively. Again, we have our crucial trade off between system performance, which can also be seen as user experienced latency and energy consumption. The evaluation is based on the two recommended modes PC/TC with active EDS (out of four possible modes), presented in Table 6.1. The reason is that disabling EDS results in permanent scanning, draining a lot of power, which is not applicable for our use case NSL.

Different parameters influence the latency and power consumption of the RE. Their individual strength varies depending on the selected mode. To have comparable values, the parameters are adapted depending on the specific mode.

6.5.1 Measurement Setup

For the measurements of both modes, we distinguish between two different scenarios, **high traffic** and **standby**.

In the first scenario, new BLE devices appear with constant frequency of once every minute. Therefore, a notification is sent every minute, containing a certain amount of devices, depending on the value of LAST_SEEN_THRESHOLD. After the start, however, the device list is empty and therefore, the first notification consists of one address only. The measurement contains the initial connection establishment. With this scenario, the power consumption of the system under a heavy workload (a lot of new devices appear) should be simulated.

In the second scenario no BLE devices appear over the whole measurement period, meaning that no new entry is added to the device list. Thus, no notification is sent. This scenario should evaluate the power consumption of the system during periods of inactivity, so where no new devices appear.

For the measurements of both modes in both scenarios, following settings are selected:

1. $\mathrm{T_{ADVI}} = 1022.5$ ms

   The RE is performing static advertising with an interval of 1022.5 ms.

2. $\mathrm{T_S} = \mathrm{t_S} = 4013.125$ ms

   The RE uses active scanning with a scan window according to the OSP policy for an advertising interval of 4000 ms. This results in a scan window of 4013.125 ms (details can be found in Chapter 4.3).

3. EDS_DURATION $= 3 \cdot t_S$

   Scanning is performed for three times the scan interval. This assures that at least every channel is scanned once.

4. Measurement duration = 10 minutes

   Each measurement takes 10 minutes and is performed three times. The standard deviation and the power consumption are calculated.

5. LAST_SEEN_THRESHOLD_MS = 70000 ms

   After a device did not send a scan request for 70 seconds it is removed from the list. Informing a smart lock about nearby devices that are not seen recently, unnecessarily consumes energy on both devices as the user is probably not in range anymore. For the scenario *high traffic* this means that except of the first notification containing one device address, all notifications are composed of two device addresses. Therefore, during the duration of one measurement (10 minutes) 10 notifications or 143 bytes of payload (19 devices) are sent.

All current consumption measurements are taken with the nRF PPK (see Section 2.2.5), supplied with a voltage of 3.303 V, using a resolution of 13 μs resulting in one sample every 13 μs. A window size of 10 samples is applied, meaning that the average of 10 data samples is logged to the output file resulting in one log entry every 130 μs.

In total three BLE devices are involved. We use one RE (nDK) and one smart lock (CDK). A third device (second nDK) is used as periodic scanner, sending scan requests to the RE.

6.5.2 Permanent Connection (PC)

The evaluation of the Permanent Connection (PC) mode with enabled EDS is described in this section.

The RE is advertising statically with an advertising interval of 1022.5 ms. After a successful device discovery that causes a new entry in the nearby device list, the RE notifies all already connected devices. Furthermore, the RE stops advertising and starts the scan process in order to discover new smart locks that are able to connect permanently to the RE. In this test, one connected smart lock was used, still scanning was active in order to look for potential other devices.

6.5.2.1 Power Consumption

The energy consumption of a BLE connection depends on two factors: the connection interval and the slave latency. The connection interval determines the frequency of connection events and thus the frequency of packet transmission between master and salve. Shorter intervals increase the energy consumption on both devices as the radio has to be turned on more frequently. The slave latency allows the slave to skip some connection intervals in order to save energy at the price of an increased latency. Therefore, we decided to use a slave latency of 0.

Three different connection intervals (4000 ms, 2000 ms, and 1000 ms) were selected in order to examine their influence on the power consumption. The measurements results of the measurement scenarios (*high traffic, standby*) are presented in Table 6.2 and Table 6.3.

The results of Table 6.2 come up with an interesting observation. Their illustration can be found in Figure 6.6a. While using a faster connection interval, the mean power

connInterval [ms]	Mean Current Consumption [mA]	Mean Power Consumption[mW]
4000	1.4848 ± 0.0503	4.9042 ± 0.1661
2000	1.0335 ± 0.0376	3.4137 ± 0.1242
1000	0.5968 ± 0.0644	1.9712 ± 0.2127

Table 6.2: *High traffic*: measured current consumption and calculated power consumption of nRF52 in PC mode over 10 minutes for different connection intervals (connInterval).

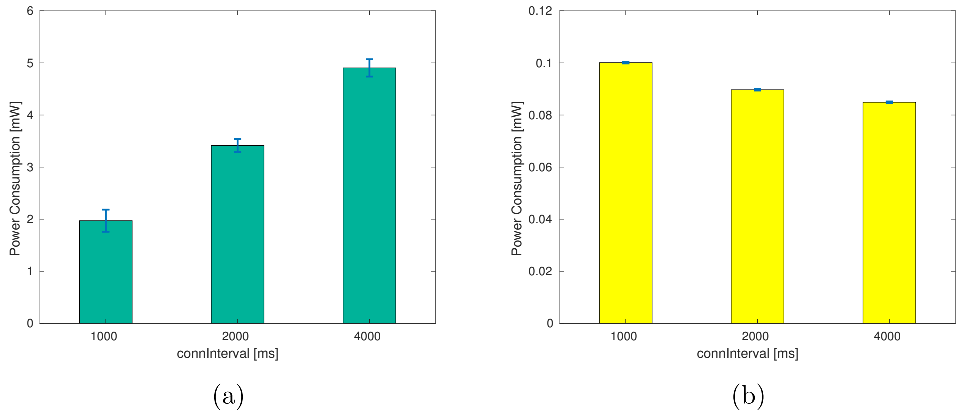


Figure 6.6: Power consumption of the RE in PC mode during *high traffic* (a) and in *standby* (b).

consumption decreases despite our expectations. The reason is the radio usage that was already discussed in Section 6.4.5.2. After each successful device discovery, the RE performs scanning in order to be able to find new smart locks which wants to get notified. Applying continuous scanning for a duration of three scan intervals would require the radio to scan for more than 12 seconds! In fact, this duration is not reached. The connection event of the existing connection terminates the scan window as the radio is needed in order to keep the connection alive. By applying a shorter connection interval, the scan window is interrupted earlier. For example, a connection interval of 1 second terminates the scan window within 1 second. The scan window is not resumed afterwards. Therefore, the radio is switched off and a lot of energy is conserved. The next scan window starts when the scan interval is over. Again, a shorter connection interval terminates the scan window earlier than a long connection interval. The exact time point of the abruption depends on the relative position of scan and connection events on the time axis. They can vary from event to event and thus they can not be determined exactly. This fact is illustrated by a relatively high standard deviation.

Activated advertising would introduce an additional event that is able to interrupt the scan window. Therefore, we highly recommend to not use advertising while scanning is active.

The influence of different connection intervals on the power consumption in the *standby* scenario is demonstrated by Table 6.3 and illustrated in Figure 6.6b. In this scenario, no other BLE devices are appearing, meaning that no notifications are necessary. This also means that the RE never scans for new smart locks that would like to get notified. There-

connInterval [ms]	Mean Current Consumption [mA]	Mean Power Consumption [mW]
4000	0.0257 ± 0.0001	0.0849 ± 0.0003
2000	0.0272 ± 0.0001	0.0897 ± 0.0003
1000	0.0303 ± 0.0001	0.1001 ± 0.0003

Table 6.3: *Standby*: measured current consumption and calculated power consumption of nRF52 in PC mode over 10 minutes for different connection intervals (connInterval). The measurement excludes the initial connection establishment.

fore, the measurements shows the energy consumption of the RE using static advertising and holding a single connection with different connection intervals. As expected, a higher connection interval increases the power consumption. In contrast to the scenario before, we can observe a very low standard deviation. This is because we have advertising and connection events only. No scanning with variable length termination is applied.

Comparing both tables shows us that the main part of the energy is consumed by the need of scanning for further devices.

6.5.2.2 Latency

Despite the power consumption, the time (L) which is needed from the point where a device enters inside the range of the RE till the successful reception of the notification by the interested device is an important indicator for the usefulness of a system. For PC, this time can be subdivided into two steps, assuming a connection is already established (see Equation 6.1):

$$L = DD_{AD,RE} + NT_{RE,NSL}(connInterval) \tag{6.1}$$

1. Device discovery of RE and appearing BLE device ($DD_{AD,RE}$)

   This time is already known. The measurement results are presented in Table 4.6. As the RE is using an advertising interval of 1022.5 ms we can assume an average device discovery time of **1078.7 ms** and a continuously scanning smart phone.

2. Sending notification from RE to interested device ($NT_{RE,NSL}$)

   The time that is needed for sending the notification depends on the selected connection interval. Data to be sent is queued until the next connection event. The time between two connection events is determined by the connection interval *connInterval*. Therefore, at most the time of the connInterval is spent here. The exact time is the time of the next connection interval minus the appearance time of a new device. As a randomization of the time of appearance of new devices was chosen, the average notification latency is close to $\frac{connInterval}{2}$. The resulting measurement values are demonstrated in Table 6.4.

The measurement results show that doubling the connection interval also doubles the average notification latency. Thus, we have **linear** relation between connection interval and notification latency.

The whole average latency can be calculated by inserting the measured and calculated values in Equation 6.1. E.g., for a connection interval of 4000 ms this would result in a total latency of 3109.7 ms.

connInterval [ms]	Notification Latency [ms]
4000	2031.0 ± 1158.8
2000	1005.5 ± 571.1
1000	522.2 ± 293.1

Table 6.4: Average notification latency in PC mode over 100 measurements, using different connection intervals.

6.5.2.3 Discussion

The Permanent Connection (PC) mode is able to maintain connections to multiple smart locks and to notify all locks quickly. Once needed, the latency can be improved by selecting a faster connection interval. This comes at the price of the multiple connection support requirement from the RE and also from the connection partner. Else the device is not discoverable for all other devices.

While supporting **multiple** smart locks, the power consumption increases due to the needed scan windows. Still, the energy consumption is acceptable, although the reason can be found in terminated scan windows. The probability of the discovery of new smart locks diminishes with every connected device. Once the smart lock is using slow advertising intervals it is unlikely to establish a working connection. While supporting only **one specific** smart lock, a very good power consumption could be achieved. In this case, scanning for further devices could be disabled and the power consumption would be close to the measurements presented in Table 6.3.

On the one hand, PC is able to sustain a high load of BLE traffic, caused by an environment that is determined by a lot of (re)appearing BLE devices. The number of appearing devices do not increase the power consumption considerably nor reduce the latency. On the other hand, in *standby* or if new devices appear very rare, holding a connection for a long time increases the power consumption unnecessarily.

Furthermore, PC mode is dependent on the environment. A lossy environment leads to disconnections. Depending on the number of currently maintained connections, the reconnection process might be time consuming. Furthermore, a high amount of reconnections consumes an unnecessary amount of energy on both devices.

6.5.3 Temporary Connection (TC)

The evaluation of the Temporary Connection (TC) mode with enabled EDS is described in this section.

The RE is using static advertising with an advertising interval of 1022.5 ms. After every received scan request, the RE stops advertising and start scanning instead. After the discovery of a smart lock, the RE initiates the connection establishment such that the content of the device list can be exchanged. After the connection establishment, the scanning process is restarted in order to look for possible other smart locks. Once the data is transmitted, the connection is disconnected by the RE.

T_{ADVI} [ms]	Mean Current Consumption [mA]	Mean Power Consumption [mW]
4000.0	2.8873 ± 0.1185	9.5369 ± 0.3914
2000.0	2.2101 ± 0.0347	7.3001 ± 0.1146
1022.5	2.1193 ± 0.1400	6.9999 ± 0.4624
417.5	1.4547 ± 0.0410	4.8050 ± 0.1354
152.5	1.371 ± 0.0463	4.5284 ± 0.1529

Table 6.5: *High traffic*: measured current consumption and calculated power consumption of nRF52 in TC mode 110 over 10 minutes for different advertising intervals (T_{ADVI}) on the connection partner.

Mean Current Consumption [mA]	Mean Power Consumption [mW]
0.0242 ± 0.0001	0.0798 ± 0.0003

Table 6.6: *Standby*: measured current consumption and calculated power consumption of nRF52 in TC mode over 10 minutes.

6.5.3.1 Power Consumption

In contrast to the PC mode, in TC mode the main influence on the power consumption is not the connection interval. The energy consumption in this mode depends on two main factors: the advertising interval of the RE (static advertising with an interval of 1022.5 ms) and the advertising interval of the device the RE has to connect to (smart locks). The reason for this dependency is the device discovery latency dependent on the advertising interval (see Table 4.6). When a smart lock is advertising with a faster interval, the RE is able to detect it faster. A faster device discovery leads to the case that the scanning process can be stopped earlier leading to the radio being powered for shorter periods.

After an exchange of information with the first device, the RE reactivates scanning to discover further smart locks. If one whole scan period (duration of EDS_DURATION) passes without the discovery of a new smart lock, the RE stops scanning. In the test scenario, the RE does not connect to more than one smart lock. Still, after the successful transmission of the device list, the RE scans for a whole period, consuming additional energy. In contrast to the first device discovery that is stopped once the advertising packet of the smart lock is received, this scan takes EDS_DURATION and is independent from the advertising interval of the potential connection partner.

In order to evaluate the influence of the connection partner's advertising interval on the power consumption of the RE, we measured the current consumption for all four available advertising intervals of the NSL (see Table 2.2) plus the advertising interval that was introduced with the introduction of AA (see Chapter 5). The results are listed in Table 6.5. As explained, longer advertising intervals of the connection partner increase the power consumption of the RE. The scan windows are the main contribution to the energy consumption and their length is determined by the device discovery latency. As this latency can vary, also the power consumption can change. Similar to PC mode, a significant reduction in power consumption can be observed when no new BLE devices are approaching. The measurement result is presented in Table 6.6. Again, this is because scanning is deactivated during the whole measurement. This means, the energy

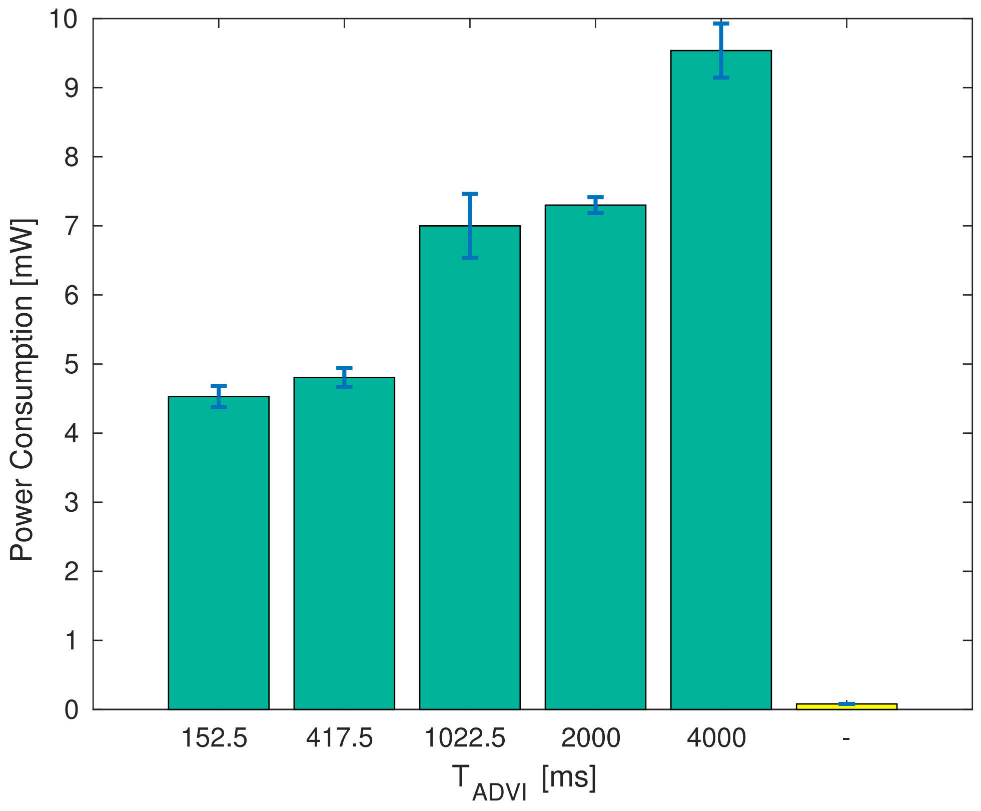


Figure 6.7: Power consumption of the RE in TC mode during *high traffic* (green) and in *standby* (yellow).

consumption consists of the energy needs of advertising only.

A comparison of both measurements is illustrated in Figure 6.7. The green bars represent the power consumption dependent on the different advertising intervals of the connection partner while new BLE devices appear periodically (*high traffic* scenario). The yellow bar illustrates the significantly lower power consumption when no new BLE devices appear and thus no scanning is needed (*standby* scenario).

6.5.3.2 Latency

Despite the power consumption, the time (L) which is needed from the point where a device enters inside the range of the RE till the successful reception of the notification by the interested device is an important indicator about the usefulness of a system. For TC, this time is composed of four different parts (see Equation 6.2):

$$L = DD_{AD,RE} + DD_{RE,NSL} + TCONN_{RE,NSL} + NT_{RE,CDK} \tag{6.2}$$

1. Device discovery of RE and appearing BLE device ($DD_{AD,RE}$)

   This time is already known. The measurement results are presented in Table 4.6. As the RE is using an advertising interval of 1022.5 ms we can assume an average

T_{ADVI} [ms]	Notification Latency [ms]
4000.0	6565.80 ± 1279.40
2000.0	3612.90 ± 599.50
1022.5	2224.90 ± 584.80
417.5	1081.50 ± 232.27
152.5	748.10 ± 173.38

Table 6.7: Average notification latency in TC mode over 100 measurements, using different advertising intervals on the remote device.

device discovery time of **1078.7 ms** and a continuously scanning smart phone.

2. Device discovery of RE and interested device ($DD_{RE,NSL}$)

   Also this time is already known. The device discovery latency is given depending on the current advertising of the smart lock (see Table 4.6). Remember that the RE is using OSP fitting an advertising interval of 4000 ms (see Chapter 4.3).

3. Connection establishment initiated by RE with interested device ($TCONN_{RE,NSL}$)

   This includes the time for connection establishment as well as the service discovery time. BLE would allow to store already discovered services and their corresponding handlers. A problem can be once an interested device wants to use information from multiple RE's. Every RE, that implements the server functionality is allowed to assign different handlers to its attributes. Therefore, for each RE handlers would need to be stored. In order to cause as less changes as possible on the interested device, the decision was made to not store already discovered parameters. Thus, on each connection establishment service discovery has to be performed.

4. Sending notification ($NT_{RE,CDK}$)

   For sending the notification, the connection interval of the connection establishment is used. This follows the ACE approach [Mik14] that uses a faster connection interval during the connection establishment process and updates the connection interval after the connection establishment was successful. Therefore, after a successful connection establishment, the notification reaches the interested device very fast.

Similar to PC mode (see Table 6.4), we measured the time between the received scan response and the successful received notification (notification latency). This measurement includes step two till four of the description above. The resulting values are listed in Table 6.7.

As expected, the device discovery latency (see Table 4.6) has direct impact on the measured notification latency (see Equation 6.2) as a connection can only be initiated once the RE discovered the interested device (smart lock). Increasing the advertising interval of the interested device increases the discovery time.

The whole average latency can be calculated by inserting the measured and calculated values in Equation 6.2. For example, when the smart lock uses an advertising interval of 4000 ms this would result in a total latency of 7644.5 ms.

6.5.3.3 Discussion

The Temporary Connection (TC) mode allows the RE and also the connection partner to support only one simultaneous connection. The connection is kept very short such that the smart lock is responsive immediately after the data transmission. Furthermore, TC mode allows a very low power consumption. During periods of inactivity (*standby*) its power consumption is determined only by its advertising interval (see Table 6.6). New smart locks are discovered quickly, also with higher advertising intervals as connection events do not disturb the scanning process.

Moreover, TC mode avoids that the smart lock wastes energy. A lossy environment handicaps the device discovery and the connection establishment, but as long no connection request is received by the smart lock, the smart lock do not even notice the presence of the RE and thus, do not consume further energy.

All its benefits come with the cost of an increased notification latency. While supporting **multiple** smart locks the notification latency is increased from lock to lock as they have to be notified consecutively. For **each** lock, device discovery, connection establishment, service discovery and notification exchange has to be performed. This increases the notification latency and makes it dependent on the advertising interval of the connection partner. Furthermore, this fact disqualifies TC mode in a *high traffic* scenario. Every (re)appearing BLE device increases the power consumption considerably as device discovery, connection establishment, and service discovery have to be performed for each lock.

6.5.4 Outcome

We introduced the concept of the RE, a BLE device, that notifies interested BLE devices about the presence of other BLE devices. The RE is implemented and evaluated in two different modes, TC and PC. This section showed an evaluation of both RE modes regarding power consumption and latency.

Figure 6.8 compares the overall latency of both modes. This means we are considering the time starting from the entrance of an approaching device into the range of the RE till the interested device received the proper notification. Based on the energy and performance measurements, we can say that PC should be used in scenarios where a relatively stable link is available or a high load is demanded. Furthermore, it is the right choice once low latency is required. As illustrated in Figure 6.8 the lowest latency can be achieved in PC mode. Furthermore, support for multiple interested devices is given when a compromise regarding the connection establishment time can be accepted. The highest possible connection interval (4000 ms) should be chosen once multiple interested devices should be supported. In contrast to that we recommend a lower connection interval like 1000 ms once only one device should be notified. The latency is decreased by paying a relatively small amount of additional energy.

TC is suited for a lossy environment with only a few of interested devices. Above all, TC is the best choice when a low load is required. With respect to the Nuki use case the TC mode is not applicable. The RE should inform the NSL about nearby devices in order to reduce its advertising interval once it is running with a slow advertising interval (to be discoverable faster for a smart phone). This means, the highest gain of the RE is when the NSL is using the slowest possible advertising interval (4000 ms). There, as illustrated in

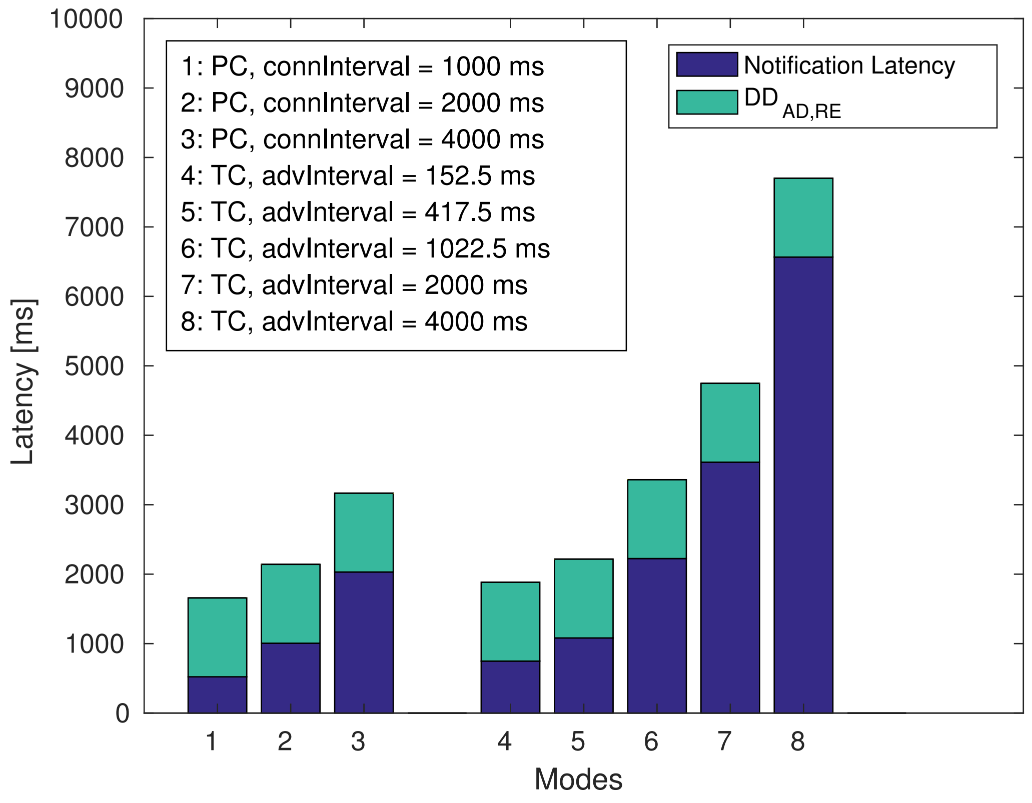


Figure 6.8: Measured average latency of the RE in the two modes, PC and TC, depending on different connection and advertising intervals. The overall latency consists of the device discovery latency between the approaching device and the RE ($DD_{AD,RE}$) and the notification latency.

Figure 6.8, the notification latency of TC mode is increased as the latency depends on the device discovery time between RE and smart lock, and thus, on the advertising interval of the smart lock. This fact could change with a longer range supported by the RE. The new BLE specification (BLE v5.0, see Section 2.1.9) increases the range of BLE significantly, making this mode interesting for this specific use case.

Once the smart lock received the notification of the RE (independent from the used mode) and resolved at least one address successfully (i.e., a trusted device is nearby), the smart lock sets its advertising interval to the fastest possible advertising interval (152.5 ms). This reduces the device discovery latency of smart phone and smart lock. Depending on the currently used advertising interval, the reduction differs (see Table 6.8). The values of the device discovery latency depending on the used advertising interval are taken from Table 4.6. In case the smart lock is currently advertising with a slower advertising interval, a **significant reduction of the device discovery latency** can be achieved (more than 95%).

Note, that on the RE an advertising interval of 1022.5 ms was chosen. Once needed, the overall latency can be reduced by choosing a faster advertising interval. The expected

T_{ADVI} [ms]	Device Discovery Latency [ms]	Reduction [%]
152.5	172.4	0
417.5	408.3	-57.8
1022.5	1078.7	-84.0
2000.0	1992.1	-91.3
4000.0	4145.8	-95.8

Table 6.8: Reduction of the device discovery latency by adapting the advertising interval to the fastest possible advertising interval (152.5 ms).

saving can be read out of Table 4.6. Of course, the power consumption of the RE is increased.

Chapter 7

Conclusion

Thanks to its low-power and low-cost properties, BLE became a popular wireless communication protocol that is nowadays supported by most consumer electronic devices, such as smart phones, laptops, and tablets. BLE is a highly versatile protocol that allows to target many different adaptation requirements. As such requirements may change over time and may be user dependent, runtime algorithms are needed in order to achieve the best performance and energy efficiency. Therefore, the major contribution of this Master's thesis are three concepts that increase the performance and energy efficiency of BLE device discovery at runtime.

First, advertising and scanning of BLE is evaluated according to its device discovery latency, energy consumption, and amount of transmitted data. We show a highly accurate mathematical model with which a prediction of the mean power consumption is possible. We demonstrate that BLE device discovery is dependent on advertising interval, scan interval, and scan window. Based on that, we introduce Optimal Scan Parameters, a mathematical model that allows to reduce the mean device discovery time by more than 4% once scan parameters are adapted accordingly.

Second, Adaptive Advertising is a smart advertising technique that adapts the advertising interval of a BLE advertiser at runtime according to a daily schedule, calculated based on user behavior. This enables a reduction of the mean device discovery latency and the energy consumption simultaneously by more than 50%. Adaptive Advertising allows to automatically adapt a BLE application to specific use cases and individual needs.

Third, the concept of a Range Extender is introduced in order to notify BLE devices about the presence of other BLE devices. Notified devices are able to adapt their parameters (e.g., advertising interval) accordingly and thus, to reduce the overall device discovery latency significantly. The Range Extender is implemented and evaluated using different parameters in two different modes: temporary connection and permanent connection, allowing to react on specific application requirements. In both modes, we are able to inform a specific BLE device on average in under two seconds about the presence of other BLE devices. We showed that depending on the used advertising intervals a reduction of the device discovery latency of more than 95% can be achieved.

Chapter 8

Future Work

Packet Reception Rate. In this thesis we assumed a packet reception rate of 100%. The two contributions Optimal Scan Parameters and Range Extender could be tested and evaluated in a lossy environment showing its influence on the mean device discovery latency and energy consumption.

Adaptive Advertising - Weekday Distinction. The current implementation of Adaptive Advertising does not distinguish between week days. The daily calculation of the schedule could be extended to a weekly calculation. This, however, requires more user data. The performance could be improved as the user experienced device discovery latency could be reduced. More user data introduces the need of data weighting, meaning that more recent time values have a stronger influence on the resulting schedule.

Adaptive Advertising - Amount of Users. From the usage data, a specific device could extract how many users are using it. Depending on the use case and the amount of identified users, the advertising interval can be adjusted further. For example, a smart lock is used by three different users and we know that all users unlocked the door in order to enter the house. Because we know that no further unlocking operation was recorded, nobody left the house. As a consequence, we know that no remaining users are outside and thus, it is not possible that the door will be accessed. Therefore, in order to save energy, we could overrule the precalculated schedule of Adaptive Advertising (which may uses a fast advertising interval) and turn advertising completely off or at least use a slower advertising interval.

Range Extender - User behavior. The time between detection and locking action may differ from user to user. For example the first user directly accesses the door while the second user drives the car into the garage and needs more time to reach the door. The second case gives the Nuki Smart Lock more time to react and a higher advertising interval could be used. Therefore, also the Range Extender could learn from its environment and adapt its advertising interval accordingly.

Range Extender - Service Discovery. After a successful service discovery, the device subscribing for notifications could store the attribute handles of the corresponding

Range Extender. After the termination of a connection, e.g., in Temporary Connection mode or after link loss, a new service discovery is not necessary anymore. Depending on the connected Range Extender, the device can use the stored attribute handle in order to subscribe for the notification service.

Range Extender - Extended Advertising. With the introduction of BLE v5.0 and its Extended Advertising (see Section 2.1.9, the two modes of the Range Extender (Temporary and Permanent) could be extended by a third, connection-less mode that uses Extended Advertising of BLE v5.0. In this mode, information about nearby devices could be broadcasted on the data channels with synchronization packets on the three advertising channels in order to inform potential interested devices. In this case, interested devices are able to omit the time consuming connection establishment and do not need to subscribe for notifications anymore. Once devices want to gain knowledge about their environment, they start listening on the advertising channels for the synchronization packets that contain information about the next used data channel. Afterwards, they switch to the data channels and receive the desired information. This mode would allow to notify multiple devices simultaneously without any delay introduced by connection establishment or other devices.

Range Extender - Longer Range. With the introduction of BLE v5.0, a longer communication range is possible. This increased range allows the Range Extender to spend more time on the notification exchange without suffering an adverse impact on the user experience latency. As now more time is available, the Permanent Connection mode could use a higher connection interval. In Temporary Connection mode the remote device could use a higher advertising interval, reducing its energy consumption.

Range Extender - Smart Scheduling of Scan and Connection Events. The Range Extender is able to support multiple connections while scanning for further potential connection partners. As demonstrated, the duration of the scan window is limited by the occurrence of connection events. To avoid early abruptions of scan windows, a smart scheduling technique for scan and connection events could be introduced.

Cooperation of Range Extender and Adaptive Advertising. Once a Range Extender is exclusively used with a device running Adaptive Advertising, this device could then inform the Range Extender about the used advertising interval. As a consequence, the Range Extender knows the advertising interval of the remote device at any point in time, allowing to adjust its scan parameters (interval and window) according to the Optimal Scan Parameters. This could reduce the mean device discovery time between both devices.

Bibliography

[Blu14] Bluetooth SIG. *Bluetooth Core Specification Version 4.2*, 12 2014. Rev. 4.2.

[Blu16] Bluetooth SIG. *Bluetooth Core Specification Version 5.0*, 12 2016. Rev. 5.0.

[Bon16] Matthew Bon. A Basic Introduction to BLE Security. Website, October 2016. `https://eewiki.net/display/Wireless/A+Basic+Introduction+to+BLE+Security`, accessed on 22.10.2017.

[BS17] Inc. Bluetooth SIG. GATT Services. Website, 2017. `https://www.bluetooth.com`, accessed on 21.11.2017.

[Chi17] Rodrigo Chiossi. Android Source. Website, September 2017. `http://androidxref.com`, accessed on 24.10.2017.

[CPH$^+$14] Keuchul Cho, Woojin Park, Moonki Hong, Gisu Park, Wooseong Cho, Jihoon Seo, and Kijun Han. Analysis of Latency Performance of Bluetooth Low Energy (BLE) Networks. *Sensors*, 15(1):59–78, 2014.

[Cyp17] Cypress Semiconductor Corporation. *PSoC 4: PSoC 4XX8_BLE Family Datasheet*, 04 2017. Rev. *L.

[DARD07] Catalin Drula, Cristiana Amza, Franck Rousseau, and Andrzej Duda. Adaptive Energy Conserving Algorithms for Neighbor Discovery in Opportunistic Bluetooth Networks. *IEEE Journal on Selected Areas in Communications*, 25(1), 2007.

[Fou17] Linux Foundation. Zephyr Project. Website, 2017. `https://www.zephyrproject.org/`, accessed on 21.11.2017.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[Gmb16] Nuki Home Solutions GmbH. Nuki Smartlock API. Website, July 2016. `https://nuki.io/wp-content/uploads/2016/04/20160731-Smartlock-API-v1.02.pdf`, accessed on 31.10.2017.

[Gmb17] Nuki Home Solutions GmbH. Nuki. Website, 2017. `https://nuki.io`, accessed on 21.11.2017.

[Gup13] Naresh Gupta. *Inside Bluetooth Low Energy.* Artech House, 2013.

[Hid17] Josu Hidalgo. Using Bluetooth on iOS. Website, May 2017. `https://www.avantica.net/blog/using-bluetooth-on-ios`, accessed on 24.10.2017.

[Inc17a] Apple Inc. Bluetooth Accessory Design Guidelines for Apple Products. Website, June 2017. `https://developer.apple.com/hardwaredrivers/BluetoothDesignGuidelines.pdf`, accessed on 24.10.2017.

[Inc17b] Monsoon Solutions Inc. Power Monitor Software. Website, 2017. `http://msoon.github.io/powermonitor`, accessed on 21.11.2017.

[JDJ17] Wha Sook Jeon, Made Harta Dwijaksara, and Dong Geun Jeong. Performance Analysis of Neighbor Discovery Process in Bluetooth Low-Energy Networks. *IEEE Transactions on Vehicular Technology*, 66(2):1865–1871, 2017.

[KL10] Sandeep Kamath and Joakim Lindh. Measuring Bluetooth Low Energy Power Consumption. *Texas Instruments Application Note AN092, Dallas*, 2010.

[KSC15] Philipp Kindt, Marco Saur, and Samarjit Chakraborty. Neighbor Discovery Latency in BLE-Like Duty-Cycled Protocols. *arXiv preprint arXiv:1509.04366*, 2015.

[KYGC15] Philipp Kindt, Daniel Yunge, Mathias Gopp, and Samarjit Chakraborty. Adaptive Online Power-Management for Bluetooth Low Energy. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 2695–2703. IEEE, 2015.

[LCM12] Jia Liu, Canfeng Chen, and Yan Ma. Modeling and Performance Analysis of Device Discovery in Bluetooth Low Energy Networks. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 1538–1543. IEEE, 2012.

[LCMX13] Jia Liu, Canfeng Chen, Yan Ma, and Ying Xu. Energy Analysis of Device Discovery for Bluetooth Low Energy. In *Vehicular Technology Conference (VTC Fall), 2013 IEEE 78th*, pages 1–5. IEEE, 2013.

[Mac67] J. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[Mik14] Konstantin Mikhaylov. Accelerated Connection Establishment (ACE) Mechanism for Bluetooth Low Energy. In *Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on*, pages 1264–1268. IEEE, 2014.

[Mon14] Monsoon Solutions, Inc. *Mobile Device Power Monitor Manual*, 05 2014. Rev. 1.14.

[Nor17a] Nordic Semiconductor. *nRF52832 Product Specification*, 10 2017. Rev. 1.4.

[Nor17b] Nordic Semiconductor. *Power Profiler Kit User Guide*, 07 2017. Rev. 1.1.

[SK16] Kannan Sadasivam and Max Kingsbury. *PSoC 4 and PSoC Analog Coprocessor Low-Power Modes and Power Reduction Techniques.* Cypress Semiconductor Corporation, 03 2016. Rev. *F.

[Spo16] Michael Spoerk. IPv6 over Bluetooth Low Energy using Contiki. Master's thesis, Graz University of Technology, 2016.