Hannah Brunner, BSc

Bringing Cross-Technology Communication to Life: Implementation of a Gateway-Free Smart Home using Off-The-Shelf Devices

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

Masterstudium Information and Computer Engineering

eingereicht an der

Technischen Universität Graz

Betreuer

Ass.Prof. Dott. Dott. mag. Dr.techn. MSc Carlo Alberto Boano

Institut für Technische Informatik

Graz, August 2020

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

Hannah Brunner, BSc

Bringing Cross-Technology Communication to Life: Implementation of a Gateway-Free Smart Home using Off-The-Shelf Devices

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dott. Dott. mag. Dr.techn. MSc Carlo Alberto Boano

Institute of Technical Informatics

Graz, August 2020

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.

Date, Signature

Kurzfassung

In den letzten Jahren ist die Anzahl der eingesetzten Geräte im Internet der Dinge stark gestiegen. Einer der Hauptgründe dieser Entwicklung ist die Verbreitung von Smart Home Technologien. Diese ermöglichen eine komfortable, drahtlose Steuerung unterschiedlichster Haushaltsgeräte, wie Lampen, Türschlössern und vielem mehr. In Smart Home Anwendungen werden verschiedene Funktechnologien eingesetzt, die jeweils unterschiedliche Anforderungen erfüllen. Die meisten Smart Home Geräte verwenden eine der drei beliebtesten und weit verbreitesten Technologien im 2.4 GHz ISM Frequenzband: Bluetooth Low Energy (BLE), ZigBee oder Wi-Fi. Obwohl diese Funktechnologien die selben Frequenzen nutzen, können Geräte mit unterschiedlichen Technologien aufgrund inkompatibler Bitübertragungsschichten nicht direkt miteinander kommunizieren. Um trotzdem einen Datenaustausch zu ermöglichen, werden Gateways eingesetzt. Gateways sind Geräte, die mehrere Funkstandards unterstützen und Daten dementsprechend in die unterschiedlichen Technologien übersetzen können. Der Einsatz von Gateways bringt jedoch mehrere Nachteile mit sich. Als Beispiele können erhöhte Kosten, gesteigerte Komplexität und zusätzlicher Datenverkehr im Netzwerk genannt werden.

In dieser Masterarbeit machen wir uns eine technologieübergreifende Kommunikationsmethode (Cross-Technology Communication - CTC) zu Nutze, um eine Smart Home Anwendung ohne die Verwendung von teuren Gateways zu entwickeln. Basierend auf dem Framework X-Burst, wird die Kommunikation zwischen Geräten inkompatibler Art ermöglicht. Im Speziellen wurde X-Burst auf einem Smartphone mit Wi-Fi Unterstützung, einer ZigBee Lampe und einem BLE Türschloss implementiert. Dadurch kann gezeigt werden, dass Gateway-freie, technologieübergreifende Kommunikation mit den wichtigsten Funktechnologien im 2.4 GHz ISM Frequenzband möglich ist. Außerdem wird ein Verfahren vorgestellt, welches die gleichzeitige Nutzung von CTC und herkömmlichen Funktechnologien, genauer ZigBee und BLE, zulässt. Diese Arbeit ist eine der ersten, die technologieübergreifende Kommunikation auf handelsüblichen und tatsächlich eingesetzten Geräten ermöglicht, zeigt das Potenzial dieser Kommunikationsmethode im Bereich des Smart Homes und ist ein erster Schritt in Richtung einer breiteren Anwendung von CTC.

Abstract

In the recent years, the number of connected devices in the Internet of Things has been rising steeply. A main driver for these advances is the adoption of smart home technology, which allows homeowners to remotely control light bulbs, blinds, door locks, and more. In smart home applications, various wireless technologies are used to satisfy different requirements. Most devices employ one of the three most popular and widespread protocols in the 2.4 GHz ISM frequency band: Bluetooth Low Energy (BLE), ZigBee, or Wi-Fi. Even though working on the same frequencies, these heterogenous devices cannot communicate with each other due to incompatible physical layers. Currently, multi-radio gateways are used to allow a data exchange despite different underlying technologies. These devices translate data from one technology to another and introduce several drawbacks, including higher costs, increased complexity, and additional network traffic.

In this thesis, we leverage Cross-Technology Communication (CTC) to build an exemplary gateway-free smart home application, where heterogeneous off-the-shelf devices can directly communicate to each other and with a smartphone. Towards this goal, the CTC framework X-Burst is implemented on a smartphone using Wi-Fi, a ZigBee light bulb, and a BLE door lock. Thereby, it can be shown that seamless bidirectional communication between the three most popular wireless technologies in the 2.4 GHz ISM frequency band is possible without the use of costly gateways. Furthermore, a coexistence mechanism is introduced to allow the concurrent use of CTC and other existing communication stacks (i.e., ZigBee and BLE). This work is one of the first to bring CTC capability on real-world devices and marks an initial steps towards a broader adoption of CTC in the Internet of Things, and in smart home applications in particular.

Danksagung

Diese Diplomarbeit wurde im Jahr 2020 am Institut für Technische Informatik an der Technischen Universität Graz durchgeführt.

Zuerst möchte ich mich bei meinem Betreuer Carlo Alberto Boano für seine stetige, hervorragende Unterstützung und das hilfreiche Feedback während der gesamten Durchführung dieser Arbeit bedanken. Weiters danke ich auch Rainer Hofmann, sowie Markus Schuß, für ihren Support und viele hilfreiche Tipps.

Ich möchte auch meiner Familie und insbesondere meinen Eltern danken, dass sie mir auf meinem Lebens- und Bildungsweg völlige Freiheit gegeben, und mich gleichzeitig jederzeit auf unterschiedlichste Weise unterstützt haben. Abschließend, danke Stephan für dein Verständnis, die hilfreichen Diskussionen und die grenzenlose Unterstützung.

Graz, im August 2020 Hannah Brunner

Acknowledgments

This Master Thesis was written during the year 2020 at the Institute of Technical Informatics at Graz University of Technology.

First and foremost, I would like to thank my supervisor Carlo Alberto Boano for his continuous and excellent support, as well as his helpful feedback throughout the entire time. Furthermore, I thank Rainer Hofmann and Markus Schuß for their support and many helpful tips.

I would also like to thank my family, especially my parents, who gave me complete freedom of choice during my entire life and education, while supporting me in every possible way. Finally, thank you Stephan for your understanding, the useful discussions and the unlimited support.

Graz, August 2020 Hannah Brunner

Contents

List of Figures

List of Tables

Abbreviations

ACK Acknowledgment.

API Application Programming Interface.

BLE Bluetooth Low Energy.

CTC Cross-Technology Communication.

HAL Hardware Abstraction Layer.

IEEE Institute of Electrical and Electronics Engineers.

IoT Internet of Things.

ISM Industrial, Scientific and Medical.

MAC Medium Access Control.

OS Operating System.

PRR Packet Reception Rate.

RAM Random Access Memory.

RDC Radio Duty Cycling.

ROM Read-Only Memory.

RSS Received Signal Strength.

SDK Software Development Kit.

SDR Software Defined Radio.

TI Texas Instruments.

UDP User Datagram Protocol.

WLAN Wireless Local Network.

WPAN Wireless Personal Network.

ZLL ZigBee Light Link.

Chapter 1

Introduction

The interest in the Internet of Things (IoT) is unbroken since it was first mentioned in 1999 [23]. It consists of connected everyday objects equipped with sensors and actuators, thus opening up a wide range of applications. Popular application examples are fitness tracking using wearables, air quality monitoring in smart cities, and the deployment of wireless objects within a smart home. The IoT plays also a major role in an industrial context: sensor data can be used to predict failures on critical infrastructure and thereby help to minimize downtime and costs [45]. The number of connected IoT devices has been rising steeply in the last decade and reached more than 9 billion in 2019. IoT Analytics [35] identified the adoption of smart home devices as one of the main drivers for the recent advances. In smart homes, common objects such as lamps, door locks, blinds or speakers, are connected wirelessly to allow remote control and make the residents' life easier. The potential of the smart home market is huge. According to [36], it is expected to reach 144 billion dollar by 2025. Especially, since big global players such as Google, Amazon, or Apple entered the stage, more and more people get familiar with smart home devices.

1.1 Problem Statement

Despite the rising sales of smart home systems, their adoption is slowed down by the increasing complexity of installation and usage [12]. Furthermore, the lack of an industry-wide communication standard leads to compatibility issues and frustration on the customers' side [11, 31, 50].

Incompatible wireless technologies. Due to a variety of different Internet of Things applications, several wireless technologies and protocols have been developed to satisfy the emerging requirements. Depending on the applications' goal and employed hardware platforms, it is necessary to exhibit certain properties (e.g., video streaming requires very high bandwidth, while small battery-operated sensors usually send only a few bytes of data but need to support low-power modes to serve a long lifetime). Typical requirements to be considered are data rate, bandwidth, power consumption, communication range, security, and price. In smart home applications, the most popular protocols are ZigBee, Z-Wave, and Bluetooth Low Energy (BLE), which are designed for low-power and low-bandwidth communication. While their data rates are clearly sufficient for controlling

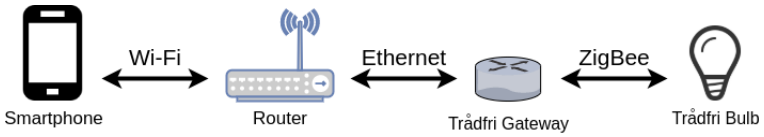


Figure 1.1: To control a Trådfri light bulb with a smartphone, two additional mains-powered devices are required: a router and a Trådfri gateway. Furthermore, three protocol translations are necessary for a simple on/off command, introducing additional network traffic and latency.

lamps, doorlocks or thermostats, they are not feasible for streaming multimedia content or security footage. A common alternative is Wi-Fi (IEEE 802.11), which features high bandwidth but is very power hungry. Apart from Z-Wave, these technologies all operate in the 2.4 GHz license-free Industrial, Scientific and Medical (ISM) band. Nevertheless, devices employing different wireless technologies, even if working in the same frequency band, cannot directly communicate with each other due to incompatible physical layers.

Gateways as alleged solution. As a smart home solely based on one technology is not feasible, multi-radio gateways are used to allow data exchange between devices with different underlying physical layers. Gateways, sometimes refered to as hubs, translate and forward data from one protocol to another. They are costly devices with high power demand and introduce a single point-of-failure. Furthermore, translation overhead and complexity increases significantly. A simple real-world example is illustrated in Fig. 1.1 and emphazises the effort introduced by the usage of multiple radios and gateways.

Lack of compatibility across different manufacturers. Another obstacle on the path to a running smart home environment are conflicting platforms. Gateways often support only devices of the same brand or selected other products. Making sure all devices are compatible with each other is a heavy burden for customers and transforms "the dream home into a nightmare"[15]. The introduction of voice-controlled gateways, such as Alexa or Google Home eased the problem, as they support a lot of devices due to their growing popularity. However, these ecosystems are still unlikely to comprehend everything [50].

No transparency on the application layer. The deployment of several different devices introduces the "app problem"[50]. Each device requires an individual app for installation and usage, as each vendor offers its own special features. Despite the inconvenience of these circumstances, the use of separate apps make a direct data exchange and control between devices impossible. Hence, a truly smart home, where devices exchange sensor data and adjust light, blinds or locks autonomously, cannot work across different vendors or without a central entity. Adding a number of devices in a smart home can require just as many gateways and apps, which soon leads to a confusing, complex and inefficient setup. [27]

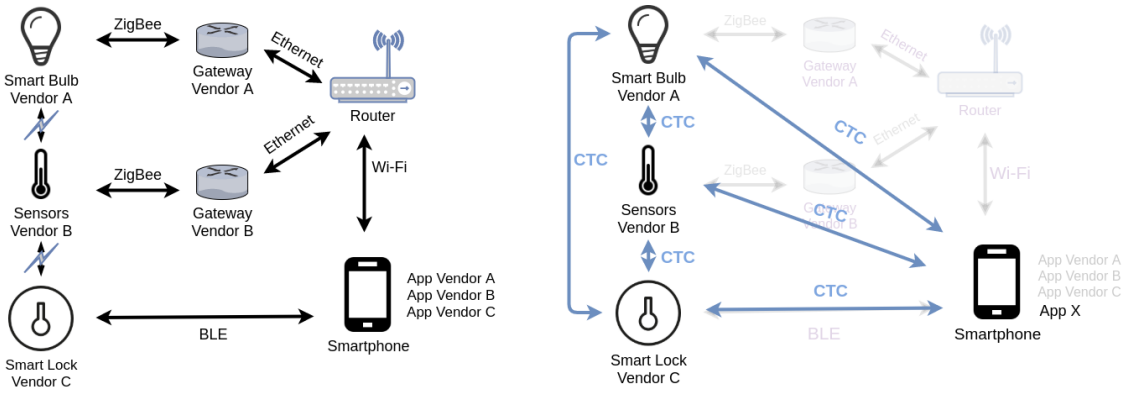


Figure 1.2: Example of a smart home without (a) and with cross-technology communication (b). In the conventional scenario (a), several gateways and many protocol translations are required. The devices cannot communicate directly and, for each device, a different app needs to be installed. The introduction of cross-technology communication allows a direct data exchange and makes gateways obsolete, as all devices can be controlled with a single app (b).

Cross-Technology Communication (CTC) can help to circumvent the aforementioned problems. It is a mechanism that allows devices employing incompatible wireless technologies to directly communicate with each other without the need of a gateway [29]. The application of CTC on smart home devices can make the use of gateways obsolete, help to save costs and simplify a smart home setup drastically, as shown in Fig. 1.2. The goal of this thesis is to show the feasibility of CTC to overcome problems introduced by the deployment of smart home devices with incompatible wireless technologies. In particalur, we want to demonstrate a gateway-free data exchange between heterogenous off-the-shelf devices and thus ultimately build a basis to enable interoperability between any smart home device, regardless of its brand or employed wireless technology.

1.2 Thesis Contributions

To enable a smart home architecture as proposed in Fig. 1.2 and to show the potential of cross-technology communication in a real-world scenario, we port and apply X-Burst [29] on several off-the-shelf smart home devices. X-Burst is a CTC framework and enables bidirectional communication between devices with incompatible wireless technologies. It has previously been showcased on ZigBee and BLE devices, but its principle is independent of the underlying technology. Data is encoded in the duration of energy bursts, which can be decoded by any device on the same channel using energy sensing. Hence, it is a suitable approach in the smart home domain, where most devices operate on overlapping channels in the 2.4 GHz ISM band.

Extension of X-Burst to support Wi-Fi devices. We apply X-Burst on two off-the-shelf Wi-Fi devices, to allow seamless and bidirectional communication between the three most popular wireless technologies in the 2.4 GHz ISM band: Wi-Fi, ZigBee and

BLE. In the context of a seminar project, we have ported X-Burst to the Raspberry Pi 3B+ already. Since smartphones are used as the main remote control for many smart home appliances, we enable CTC support on the Wi-Fi chip `bcm4358` of the Nexus 6P in this thesis.

Applying X-Burst on off-the-shelf smart home devices. We show the feasibility of CTC on popular real-world smart home devices. In particular, we can control a ZigBee-based light bulb and a BLE-based doorlock with a smartphone's Wi-Fi radio. Towards this goal, X-Burst is ported to the IKEA Trådfri, a smart light bulb employing a ZigBee radio. Further, X-Burst is ported to a BLE-enabled door lock, the Danalock V3.

Seamless coexistence of CTC with the existing communication stacks. We implement the CTC functionality such that BLE and ZigBee devices can use their native communication mechanisms and CTC simultaneously. In other words, all smart home devices can still be controlled with their original remote controllers, while additional data exchange with any other CTC-enabled device is now possible.

Building a concrete smart home demonstrator making use of CTC. We build a demonstrator to show that a gateway-free smart home is possible despite incompatibilities of the employed wireless technologies. In particular, the demonstrator allows to control a BLE-enabled door lock and a ZigBee-enabled light bulb using a Wi-Fi based smartphone from one single app.

1.3 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 contains an overview of the three most popular wireless technologies in the 2.4 GHz ISM band, which have been considered in this thesis. Furthermore, it introduces the reader to cross-technology communication and, in particular, to the CTC framework X-Burst. At the end of the chapter, the employed hardware and software platforms are described briefly. In Chapter 3, current approaches to enable communication between incompatible wireless devices are presented. First, smart home gateway solutions are described, followed by a summary of existing work on CTC. Chapter 4 identifies the requirements to enable CTC on Wi-Fi devices, i.e., how to port X-Burst on the Raspberry Pi 3B+ and the Nexus 6P. Chapter 5 is devoted to the design of a gateway-free smart home application. The necessary implementation steps to allow seamless cross-technology communication between the Nexus 6P smartphone, the IKEA Trådfri light bulb, and the Danalock V3 door lock are described. We perform an experimental evaluation on all devices and show its results in Chapter 6. The evaluation contains information about the achievable throughput, the communication range and the reception performance in the presence of external radio interference. Furthermore, the seamless coexistence between CTC and the existing communication stacks is evaluated. Finally, we conclude the thesis in Chapter 7 and give an outlook on future work.

Chapter 2

Background

This chapter contains a description of the three most popular wireless technologies in the 2.4 GHz frequency band that have been considered in this thesis in Sec. 2.1. Furthermore, the basic idea and concepts of cross-technology communication are explained in Sec. 2.2, followed by a more detailed discussion on the CTC mechanism X-Burst in Sec. 2.3. Finally, an overview of the employed hardware platforms and software is given in Sec. 2.4 and Sec. 2.5, respectively.

2.1 Main Wireless Technologies Operating in the 2.4 GHz ISM Band

The Industrial, Scientific and Medical (ISM) bands are unlicensed, hence freely usable parts of the frequency spectrum. Especially the 2.4 GHz ISM band is very popular due to its global availability. It is the favored target for cross-technology communication schemes, as many different wireless technologies operate on overlapping channels within these bands. The most dominating wireless technologies used in smart home applications work also in these frequencies. In the following section, we give an overview of the three most widespread wireless technologies in the 2.4 GHz band, namely ZigBee, Bluetooth Low Energy, and Wi-Fi.

2.1.1 ZigBee

ZigBee is a standard for cost-effective low power wireless communication. ZigBee defines the behavior of wireless devices from radio to application layer to provide robust and reliable communication along with interoperability across different vendors. It is managed by the ZigBee Alliance [14], consisting of more than 200 companies trying to standardize the communication in different domains. ZigBee is especially popular in smart home applications, due to its low power-consumption and smart-home tailored application profiles.

Physical Layer

ZigBee is based on the IEEE 802.15.4 radio specification. IEEE 802.15.4 is a standard for Low-Rate Wireless Personal Area Networks (LR-WPAN). It defines the physical layer and the Medium Access Control (MAC) layer of ZigBee's network stack. ZigBee supports

different network topologies such as star and mesh topology, and can operate in several ISM frequency bands (868 MHz, 915 MHz and 2.4 GHz). It is commonly used in the 2.4 GHz band, as it is the only one that is freely available world-wide. In this band, the IEEE 802.15.4 specification defines 16 possible channels (11-26), spaced 5 MHz apart, with a bandwidth of 2 MHz each. The communication is based on the Direct-Sequence Spread Spectrum (DSSS) technique and allows a data rate of up to 250 kbit/s and a range of 10 - 100 m. The physical layer further supports sleep modes, which is especially of interest for battery driven IoT devices with small energy budget [33].

Device Types

ZigBee devices can be configured as one of the three following device types:

- Coordinator
- Router
- End Device

The *coordinator* is a device which starts and controls the network. It selects operational parameters (e.g., channel, network identifier) and stores network information. A *router* is a device which can relay messages from other devices and thus extends the network area coverage. *End devices* cannot perform any routing operations but can only talk to their parents [20].

ZigBee Cluster Library (ZCL)

ZigBee defines several application profiles to allow interoperability for devices employed in specific application domains. Among others, there are application profiles for Home Automation (HA), Smart Energy (SE) and Smart Lighting (ZLL). Application profiles specify the messaging scheme between devices in a specific application and are based on the ZigBee Cluster Library (ZCL). The ZCL is a library of clusters, containing a set of attributes and messages, corresponding to the required functionality. The clusters can be reused to add desired capabilities to an application. Examples for clusters are shown in Table 2.1 [40].

ZigBee Light Link (ZLL)

ZigBee Light Link (ZLL) is an application profile tailored for smart lighting. It aims to provide a user-friendly and intuitive installation procedure and has been introduced to maximize the interoperability between products of different smart light manufacturers. ZLL includes several clusters defined in the ZCL, which are required to control smart lighting devices, such as bulbs, remotes, switches or sensors.

ZLL further features a 'ZLL Commissioning' cluster enabling *Touchlink* installation of ZLL devices. *Touchlink* allows network commissioning without the need of a coordinator device and should thereby simplify the installation and configuration procedure for customers. Commonly, switches and remotes are configured as so-called 'initiators' which want to connect to light bulbs ('targets'). During a *Touchlink* pairing process, the devices are hold very close to each other. The initiator starts the commissioning (e.g., by pressing a button) and retrieves network parameters from the target. To complete the pairing

process, the initiator requests the target to form a new network or join an existing one. Hence, a remote can connect to several bulbs simultaneously in an easy and quick way [39].

Cluster	Description
`Basic`	This cluster contains basic information of the ZigBee device and allows to set user-defined properties.
`Time`	This cluster provides access to the real-time clock of a ZigBee device.
`On/Off`	This cluster includes messages to set a device into 'on' or 'off' state, respectively.
`Level Control`	This cluster allows to set the level of a device's physical quantity.

Table 2.1: Examples of clusters defined in the ZigBee Cluster Library (ZCL).

2.1.2 Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) is a wireless communication standard developed for short-range communication. It is part of the Bluetooth 4.0 specification and, compared to Bluetooth Classic, especially designed for low-power applications on battery-operated devices. Although covered in the same specification, BLE and Bluetooth Classic (the well-known technology for e.g., wireless headphones) are not compatible with each other. Despite BLE's rather young age (it has been introduced in 2010), it is wide-spread and highly adopted in modern consumer devices such as smartphones and notebooks. This fact, combined with the low costs of integrated BLE chips, makes it a popular wireless technology for many IoT applications [49].

The BLE stack implementation is separated into two main parts: the *Controller* and the *Host*. The *Controller* includes the radio specification consisting of physical layer and link layer, while the *Host* contains upper layer functionality [46].

Physical Layer

BLE devices operate in the 2.4 GHz frequency band on 40 different channels. Each channel has a 2 MHz bandwidth and spacing, i.e., the channels are adjacent. Three of these channels (37, 38, 39) are reserved for advertisement messages as explained in further detail below. The remaining channels can be used for bidirectional data exchange. The applied modulation scheme, called Gaussian Frequency Shift Keying (GFSK), allows a data rate of up to 1 Mbps. BLE devices typically have a range of a few tens of meters.

Device Types

BLE supports two different types of communication: *connection-less broadcasting* and bidirectional *connection-based communication.*

Using *connection-less broadcasting*, two separate device roles are specified:

- Broadcaster;
- Observer.

A *Broadcaster* can transmit non-connectable packets periodically to several devices simultaneously. *Oberservers* are devices that repeatedly scan the advertising channels to receive broadcast messages. Broadcasting can only be used for small amounts of data and does not feature any security provisions.

Connection-based communication involves two different device types:

- Central;
- Peripheral.

A *Peripheral* is commonly a small and battery-powered device (e.g., a sensor), which can connect to a single Central device at a given time. Therefore, a Peripheral transmits connectable advertising packets on predefined channels (37, 38, 39) to indicate its presence.

A *Central* is typically more powerful (e.g., smartphone, tablet) and can connect to several Peripherals. It constantly scans the advertising channels for packets and, if desired, establishes a connection with the Peripheral. Once they are connected, the Central is responsible for timing and can initiate a bidirectional data exchange periodically, i.e., the Peripheral can only respond after receiving a request from the Central device. The *connection interval* defines the time between two data transfers and is configurable between 7.5 to 4000 ms [49].

Generic Attribute Profile (GATT)

The Generic Attribute Profile (GATT) specifies how two BLE devices exchange application specific data. In GATT, data is organized in a hierarchical way using services and characteristics. Services combine several related characteristics into a logical block. A characteristic describes a single value with a well-defined format. For example, a humidity service may include a temperature characteristic as well as a humidity characteristic. As GATT offers many standard services and characteristics, it allows to easily create new applications and provides interoperability among different vendors, similar to the ZigBee Cluster Library (ZCL) in ZigBee.

2.1.3 Wi-Fi

Wi-Fi is a wireless communication technology for Wireless Local Area Networks (WLAN) specified in the IEEE 802.11 standard. It is highly adopted and commonly used to connect devices such as notebooks, smartphones and tablets to the Internet. In contrast to BLE and ZigBee, focusing on low-power communication, Wi-Fi is optimized for high-throughput data transfer. As high data rates imply high energy requirements, Wi-Fi is often not suitable for constrained battery-powered IoT devices. It is considered a valid choice in the smart home domain, however, as many smart home appliances are mains-powered by default (e.g., light bulbs) or have high bandwidth demands (e.g., security camera). Furthermore, the prevelance of Wi-Fi access points in customers' homes is beneficial, as installation effort and costs can be kept low.

Physical Layer

The main bands used for Wi-Fi communication are the 2.4 GHz and 5 GHz band. In IEEE 802.11, 14 different channels are defined in the 2.4 GHz band, of which only 13 are allowed to use in Europe. The channels have a bandwidth of 22 MHz each and are separated 5 MHz apart from each other. It is common to use the most popular channels 1, 6 and 11 only, as they do not overlap. Depending on the Wi-Fi version, different modulation schemes are applied and data rates of several tens of Mbit/s can be achieved.

2.2 Cross-Technology Communication

Cross-Technology Communication (CTC) allows devices with heterogeneous wireless technologies to directly communicate with each other. The possibility to exchange data across different technologies can help to mitigate problems arising in the crowded 2.4 GHz ISM band. CTC can be used to apply channel coordination schemes to minimize cross-technology interference [51]. Further, the deployment of multi-radio gateways can be avoided, which reduces network traffic and channel congestion. Another application of CTC is clock synchronization of incompatible devices [52].

Existing CTC techniques can be divided into two categories: packet-level modulation and physical-layer emulation.

In *physical-layer emulation*, a packet of one technology is embedded within the payload of another. For instance, the parts of a Wi-Fi frame are adjusted such that this portion resembles a ZigBee packet and can hence be decoded by commodity ZigBee receivers [34]. While this technology-specific approach enables cross-technology transmission with very high data rates, it does not allow for bidirectional communication, as only the high-end transmitter can send to a low-end receiver.

Packet-level modulation leverages packet-level information such as transmission power, duration, or interval of standard-compliant packets to encode information. Almost any wireless device on the same channel, regardless of the underlying technology, can extract and decode the corresponding data by means of energy sensing. Energy sensing is typically used for Clear Channel Assessment (CCA), i.e., to determine if a channel is occupied before transmission. Therefore, almost all radios provide this feature, which makes packt-level modulation a very generic approach. Packet-level modulation features two types of encoding: energy modulation, where information is encoded in the energy of the packets (transmission power), and time modulation, where information is encoded in the timing of the packets (duration, interval). Due to sparse packet-level information and low sampling rates, physical-layer emulation outperforms the packet-level modulation mechanism in terms of achievable data rates. However, packet-level modulation is not limited to a specific technology and thus allows bidirectional communication between various communication standards. Furthermore, broadcast cross-technology frames can be sent, i.e., it is possible to transmit information to devices making use of heterogeneous technologies simultaneously. This cannot be done using physical-layer emulation.

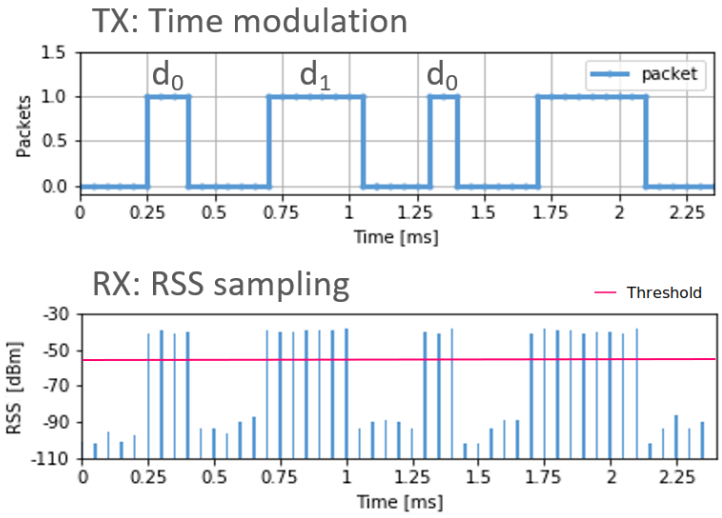


Figure 2.1: Principle of X-Burst's CTC mechanism. Transmitters translate data into predefined durations and transmit legitimate packets of corresponding length. Receivers detect the packets by continuously sampling the energy level (i.e., using RSS sampling) and extract the durations by comparing the retrieved RSS values against a predefined threshold. The durations are translated back to data.

2.3 X-Burst

X-Burst [29] is a generic and portable CTC framework based on packet-level modulation. Compared to other approaches implemented on high-end transceivers or Software Defined Radios (SDRs) [19, 53], X-Burst is applicable on off-the-shelf devices with severe constraints in processing and sampling rates. Integrated into Contiki OS, it can easily be ported to various hardware platforms with heterogeneous characteristics. X-Burst has been previously showcased on the TI CC2650 LaunchPad (BLE and ZigBee), Zolertia Firefly (ZigBee) and TelosB mote (ZigBee). Its principle is not limited to BLE and ZigBee, and thus can be applied on any wireless technology operating in the same frequency bands (e.g., Wi-Fi). Further, it allows seamless bidirectional communication and data rates up to 5 kBit/s, depending on the radio characteristics of the participating devices.

2.3.1 Concept

The working principle of X-Burst, as suggested by the name, is based on the transmission of timely encoded energy bursts (see Fig. 2.1). On the transmitter side, energy bursts are generated by sending legitimate data packets. The receiver, operating on the same channel, can detect those bursts by continuously monitoring the energy level. The data is encoded in the duration of the burst, i.e., several bits of information are mapped to a predefined duration. Hence, arbitrary data can be translated into durations and transmitted as a successive stream of energy bursts. The length of energy bursts is controlled by adjusting the payload length accordingly. Any receiver can extract the durations of the bursts by monitoring the energy level on the channel by means of RSS sampling. The Received

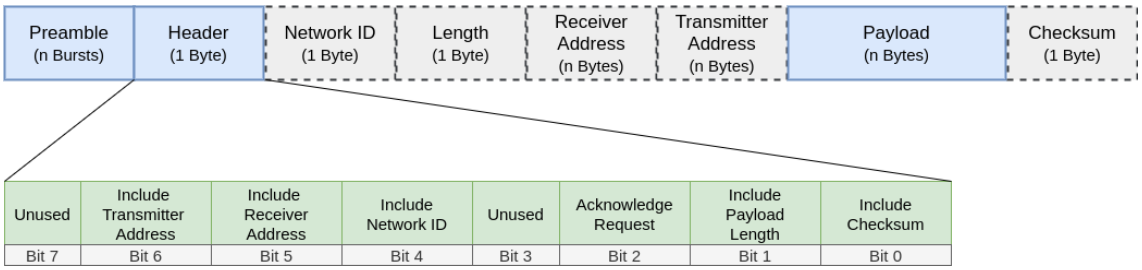


Figure 2.2: Frame format of X-Burst messages. A message must at least contain a predefined preamble, a 1-byte header and payload of arbitrary length. The header byte is a bitmask containing information about the composition of the CTC message. The remaining fields (colored in grey) are optional.

Signal Strength (RSS) is a parameter representing the energy present on a given channel and is obtained by means of Energy Detection (ED). Once the retrieved RSS value exceeds a certain threshold, the beginning of a burst is detected and its duration measured and stored accordingly. Finally, the sequence of stored durations can be retranslated into data.

Frame Format

X-Burst messages are composed of a preamble, a header and an arbitrary sized payload. It is further possible to embed additional information within the message according to X-Burst's frame format shown in Fig. 2.2. Its fields are described in more detail below.

Preamble

Each CTC message must be preceded by a preamble to allow receivers to distinguish CTC traffic from background noise. The preamble, a predefined series of bursts, is configurable and must be shared among all participating devices prior to CTC communication. After detection of a preamble, receivers start to extract and store the durations of energy bursts on the channel. If no more bursts can be detected, i.e., if the pause between two successive bursts exceeds a predefined value, the end of the CTC message is assumed and the stored durations are processed accordingly.

Header

The 1-byte header is a bitmask containing information about the composition of the CTC message. If the corresponding bit is set, the desired additional field (e.g., checksum or address information) is appended to the header in a fixed order. If the *Acknowledge Request*-bit is set, no additional data is transmitted, but the receiver is required to send an *Acknowledgement Frame* upon message reception.

Network ID, Receiver Address, Transmitter Address

These optional fields contain addressing information and can be used to direct messages to specific receivers. They allow a distinction between broadcast and unicast messages

and to assign devices to different networks. The network ID is one byte long, the length of receiver and transmitter address is configurable.

Length

The length byte is optional and indicates the number of bytes transmitted in the payload. It allows the receiver to discard falsely detected bursts after the expected number of bursts has been received.

Payload

The payload field contains the actual data. Its length can be chosen arbitrary.

Checksum

The optional checksum field allows the detection of transmission errors. It is one byte long and simply represents the sum computed over all payload bytes. After the reception of a CTC message containing the checksum, the receiver can calculate the checksum of the received payload and compares it with the checksum contained in the message. If they are not identical, the message is considered invalid and is discarded.

2.3.2 Requirements

X-Burst's CTC mechanism is not bound to a specific wireless technology or hardware platform. However, a device has to fulfill the following requirements to qualify for the deployment of X-Burst.

Shared Frequency Band and Overlapping Channels

Participating devices have to operate in the same frequency band and agree on a common channel. While a partial overlap of the selected channel is sufficient for communication, a complete overlap proves to be more robust. Previous work on X-Burst and this thesis focus on communication in the 2.4 GHz ISM frequency band, as it is the most common target for IoT devices. In principle, however, X-Burst is not limited to this portion of the spectrum.

Continuous RSS Sampling

To receive CTC messages, a device must be able to detect energy bursts by means of RSS sampling. In order to reliable determine the duration of an energy burst, a minimum sampling rate of RSS measurements is required. The sampling rate influences the minimal detectable duration as well as the minimal difference between two durations, which can be distinguished properly. Generally, the Nyquist criteria applies, i.e., the sampling interval cannot be larger than half the minimum duration and half the minimum difference between durations, respectively. Furthermore, the RSS sampling rate affects the achievable throughput as further described in Sec. 2.3.4

Transmission of Packets with arbitrary Length

A CTC transmitter has to be able to generate energy bursts of different length. Hence, the ability to send packets with varying payload length is required. The minimum and maximum duration of a burst (i.e., the minimum/maximum length of a packet) depends on the used communication standard, data rate, and payload and affects the chosen alphabet as described below.

Agreement on a Common Alphabet

In order to allow a seamless and bidirectional communication, all devices have to agree on a common alphabet. The alphabet is the mapping of symbols to a predefined set of durations. It strongly depends on the hardware characteristics of the participating devices (e.g., timer resolution, RSS sampling frequency) and the employed wireless technology (the minimum/maximum burst duration and the granularity of a duration is determined by the radio standard). In X-Burst, the alphabet size is configurable. It is given by 2^n with $n = \{1, 2, 4\}$. For example, in a 2-bit mapping ($n = 2$), two bit of information are mapped to one duration. Hence, a set of $2^2 = 4$ distinct durations is required to allow the encoding of one byte and four consecutive bursts have to be sent to transmit one byte of data.

An alphabet, which satisfies the constraints of all involved devices, can be computed using the following rules [29]:

- Across the minimum burst durations (i.e., shortest packet which can be generated) of all participating devices, choose the longest as the minimum duration (d_0) used in the alphabet.
- Across the maximum burst durations (i.e., longest packet which can be generated) of all participating devices, choose the shortest as the maximum duration (d_{max}) used in the alphabet.
- For each device, compute the minimum spacing (s) between two burst durations, which can still be detected reliably by all devices. s depends on the RSS sampling interval and must fulfill the Nyquist criteria ($s = 2 \cdot t_{sampling}$).
- The largest s among all devices (s_{max}) can be further used to compute the final set of durations, with $d_i = d_{i-1} + s_{max}$, with $i = 1, \ldots, 2^n$.
- Verify if each $d_i \leq d_{max}$, else recalculate the set of durations with a different mapping. E.g., 4-bit mapping requires 16 distinct durations, some of which can be very long.

2.3.3 Modular Architecture

A main advantage of X-Burst is its modular and generic design. Compared to other CTC mechanisms, which are implemented on specific hardware platforms only, it is integrated into the Contiki operating system and therefore easily portable to a mass of devices. Furthermore, its modular architecture, shown in Fig. 2.3, makes X-Burst highly configurable and thus, allows adaptation to heterogeneous hardware capabilities.

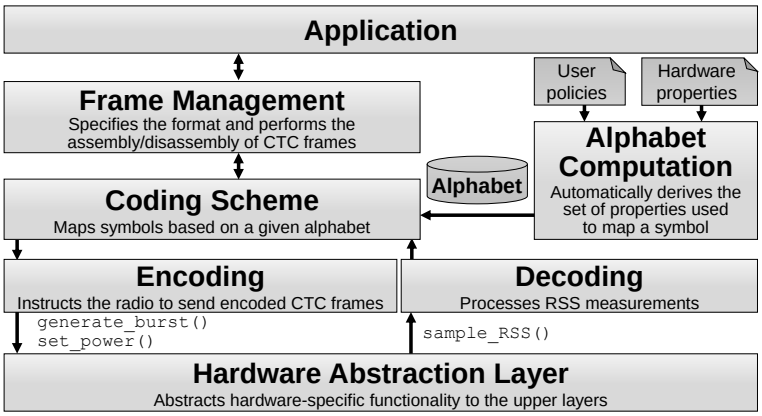


Figure 2.3: X-Burst's modular architecture [17].

Hardware Abstraction Layer

A Hardware Abstraction Layer (HAL) separates platform-specific implementation details from the actual CTC functionality. It is the only layer which has direct access to radio functionality and is thus required to keep the CTC implementation hardware-agnostic. The HAL layer has to be realized for each device individually and provides functions to perform radio initialization, RSS sampling, and to generate bursts of a certain length.

Encoding Module

The encoding module uses primitives offered by the HAL to instruct the radio to transmit energy bursts of different length. The simplest and most robust approach is to encode the data in the length of durations only, hence this module simply forwards the durations to the lower layer (`burst-only`). Another possibility is to additionally exploit the gaps between two bursts to encode data. In this case, the encoding module is responsible for delaying the transmission of successive bursts accordingly (`burst-and-gap`). Due to hardware constraints, we consider just the `burst-only` coding scheme in this thesis.

Decoding Module

The decoding module retrieves RSS measurements from the underlying HAL layer and is responsible for extracting the length of the energy bursts (and, if required, the duration of the gap between two bursts).

Coding Scheme

In this module, the selected alphabet, as described in Sec. 2.3.2, is applied, i.e., the data is translated into burst durations (`burst-only`) or burst and gap durations (`burst-and-gap`) and forwarded to the encoding module. Conversely, during reception, the extracted burst lengths are remapped to data.

Frame Management

Finally, CTC messages are assembled and disassembled in the Frame Management layer, which includes at least the creation/processing of a preamble, a 1-byte header and the payload. Further options, such as the addition of a checksum or addresses are possible as explained in Sec. 2.3.2 in more detail.

2.3.4 Achievable Throughput

Using the `burst-only` coding scheme, X-Burst allows a throughput of up to 3 kbit/s [29]. The throughput depends on hardware properties of the participating devices and the applied alphabet.

Since data transmission is based on bursts with different length, the predefined set of durations influences the throughput directly. High processing speeds and RSS sampling rates allow an alphabet composed of short durations, which increases the achievable throughput. The latter further depends on the actual content of the CTC message, as higher values (e.g., '0xFF') are translated into longer durations than low values (e.g., '0x00') and are thus transmitted slower. The actual achievable throughput cannot be determined without prior knowledge of the data, but only an upper and lower bound can be given.

Some device-specific characteristics affect the possible throughput indirectly for all devices (e.g., the RSS sampling rate has an impact on the chosen alphabet). The radio's *preparation time*, instead, has an immediate impact on the transmission speed of each device individually. The *preparation time* defines the required time between the generation of two successive energy bursts. It varies depending on the used hardware and hence, the devices' possible throughput differs even if a common alphabet is used. A certain minimum *preparation time* is necessary to allow the distinction between consecutive bursts. High values, however, decrease the achievable throughput significantly.

2.4 Employed Hardware

In the following section, the hardware used throughout this thesis is described briefly. The devices, ranging from real-world smart home devices to radio-specific development kits, are categorized into their employed wireless technologies.

2.4.1 Wi-Fi Devices

Smartphone Nexus 6P

The Huawei Nexus 6P [2] is a smartphone developed by Google and was released in 2015. It features a 5.7" display, up to 128 GB of internal memory and a 12.3 MP camera. It supports the Android 8.0 (Oreo) operating system, running on an octa-core Cortex-A57. The Nexus 6P offers the most common sensors (e.g., fingerprint sensor, accelerometer, barometer) and communication interfaces (e.g., Bluetooth, NFC, USB, GPS). Furthermore, it employs a `bcm4358` Broadcom Wi-Fi chip, which allows the implementation of X-Burst as explained in Chap. 4 and Sec. 5.2, respectively.

Raspberry Pi 3B+

The Raspberry Pi 3B+ [8] is the third generation of the Raspberry Pi series of single-board computers and was released in 2018. It is popular due to its low cost (currently 35$) and small size. Featuring a 1.4 GHz 64-bit quad-core ARM Cortex-A53 CPU and 1 GB of RAM, it offers, among others, interfaces for USB, Ethernet, Bluetooth and HDMI. A 40 pin GPIO header allows the connection and control of further devices (e.g., a display or sensors). Typically, it is used with the recommended Debian-based operating system called Raspberry Pi OS (formerly known as Raspbian). However, there is a variety of alternative operating systems available.

As the Raspberry Pi 3B+ employs a `bcm4355c0` Broadcom Wi-Fi chip, it has been previously used to enable X-Burst on Wi-Fi devices (see Chap. 4) and to even generate controllable RF interference [43].

2.4.2 ZigBee Devices

IKEA Trådfri Bulb

In this thesis, we use the IKEA Trådfri GU10 light bulb [3]. It is part of the popular IKEA Trådfri product family, which aims to bring affordable smart home solutions into people's homes and includes different smart gadgets, such as lighting, outlets, and blinds. The Trådfri bulb can be turned on, switched off or dimmed remotely using an IKEA Trådfri remote control or, if an IKEA Trådfri gateway is present, a smartphone running the IKEA app. The IKEA app allows to assign several light sources to specific groups to enable smart control based on the users' needs. The wireless communication between the Trådfri bulb and the corresponding remote/gateway is based on ZigBee. Up to ten devices can be connected to a single remote using the ZigBee *Touchlink* pairing procedure.

The IKEA Trådfri is also a popular target for DIY enthusiasts, as it has been cracked open previously and allows to flash custom firmware. According to [47], the Trådfri employs a powerful Silicon Labs EFR32MG1 Wireless Mighty Gecko SoC, integrating a ZigBee radio feasible to implement X-Burst.

Silicon Labs Thunderboard Sense

The Thunderboard Sense [44] is a compact low-power development platform for wireless sensor nodes based on the EFR32 Mighty Gecko SoC. The board contains six different environmental sensors, RGB LEDs and several buttons. It further features an on-board J-Link debugger to allow easy programming via USB, breakout pads for connection to external hardware and a CR2032 coin cell connector. The board's core is the EFR32MG SoC, integrating an ARM Cortex M4 CPU (with 256 kB flash and 32 kB RAM) and a 2.4 GHz radio module. The radio module allows the implementation of multiprotocol applications including ZigBee, BLE and proprietary protocols. The Thunderboard Sense can be used with different operating systems (e.g., Contiki-NG). Typically, however, the employed software is based on SDKs offered by Silicon Labs, as explained in Sec. 2.5.3.

Both the Trådfri bulb and the Thunderboard Sense employ the same EFR32 SoC, which allows us to use the latter as a development kit for the CTC implementation on the

EFR32, offering a more convenient programming and debugging experience compared to the bulb.

Zolertia Firefly

The Firefly [54] is a breakout board from Zolertia designed for the development of IoT applications. Besides two buttons, one RGB LED, two hardware encryption engines and an on-board USB-to-UART bridge to allow fast and easy programming, the board is equipped with a CC2538 ARM Cortex-M3 (512 kB flash, 32 kB RAM) and two radio transceivers operating on the 868 MHz, 915 MHz, and 2.4 GHz ISM bands. Furthermore, the Firefly is a ZigBee-compliant device and fully supported in Contiki OS.

The Firefly board has been previously used in [29] to showcase the functionality of X-Burst. In this thesis it is used as reference platform in the evaluation presented in Chap. 6.

2.4.3 BLE Devices

Danalock

The Danalock V3 [21] is a wireless smart lock developed by the Danish company Danalock International ApS. It can be controlled with the smartphone via BLE and is compatible with standard doors. Danalock offers an app which allows to grant access to the user's home for family members, friends and guests. Security is provided thanks to AES 256 encryption. The lock is battery-powered and features a lifetime of more than 12 months depending on daily use.

According to [38], the Danalock V3 employs a Nordic nRF52832 Bluetooth 5.2 SoC, combining an ARM Cortex M4 32-bit processor with 512 kB flash and 64 kB RAM and a 2.4 GHz BLE compliant radio transceiver.

Nordic Semiconductor nRF52840 DK

The nRF52840 DK is a single board development kit for wireless communication in the 2.4 GHz band on the nRF52840 SoC. It supports multiple protocols (e.g., BLE, ZigBee, Thread, IEEE 802.15.4) and is equipped with plenty of components such as LEDs, buttons, a NFC antenna, external memory and more. As Nordic Semiconductor offers a lot of prebuild firmware binaries and application examples, it is a versatile tool to develop IoT applications. Nordic Semiconductor further provides the nRF5 SDK, which is required to program the nRF52840 DK and is explained in more detail in Sec. 2.5.4.

Compared to the nRF52832 employed in the Danalock, the nRF52840 provides more memory (1 MB flash and 256 kB RAM) and supports more wireless protocols. Still, the devices behave similar and the software is based on the same SDK; hence the nRF52840 can be used for the development of CTC.

TI CC2650 LaunchPad

The TI CC2650 LaunchPad [48] is a development kit for the ultra-low-power CC2650 wireless MCU, integrating a 32-bit ARM Cortex M3 processor and a 2.4 GHz RF transceiver compatible with BLE 4.2 and IEEE 802.15.4. The MCU embeds 128 kB of flash and 20 kB

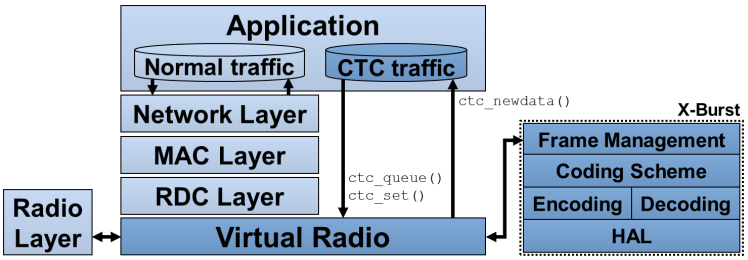


Figure 2.4: Contiki's network stack including the integration of X-Burst, taken from [29]. The network stack consists of a radio layer, RDC and MAC layer as well as a network layer on top. X-Burst is integrated using a *virtual radio* to schedule CTC operations next to normal network traffic.

of RAM, and provides extremely low power-consumption, due to the support of low-power modes. Similar to other development boards, the TI CC2650 LaunchPad features buttons, LEDs and a convenient USB programming interface.

The LaunchPad can be used with Contiki OS and was the first hardware platform to support X-Burst [28]. Equally to the Firefly, we use the TI CC2650 LaunchPad for evaluation purposes.

2.5 Employed Software

The following section contains information about the software used on the aforementioned devices. We describe the Contiki OS, on which X-Burst has been first implemented, and Nexmon, a patching framework required to extract CTC-related features on Wi-Fi devices. Finally, an overview of the SDKs used for development on the EFR32 and nRF52 SoCs is given.

2.5.1 Contiki

Contiki [22] is a lightweight open-source operating system for IoT devices. It has been developed to run on tiny low-power, memory-constrained microcontrollers and supports a variety of different platforms. Implemented in C, it is based on an event-driven kernel and aims to allow the development of hardware independent IoT applications. As Contiki is targeted for wireless devices, it features a network stack including several standard low-power protocols such as IPv6 over Low power Wireless Personal Area Networks (6LoWPAN), Routing Protocol for Low power and Lossy Networks (RPL) and the Constrained Application Protocol (CoAP).

Contiki's network stack consists of four layers shown in Fig. 2.4 in light blue. The *radio layer* contains the implementation of platform-specific radio functionalities (e.g., the transmission and reception of packets). The *RDC layer* includes different duty-cycling protocols to enable energy-saving network operations. The *MAC layer* is responsible for organizing access to the radio channel to prevent collisions and transmission errors. Finally, the *network layer* provides addressing and routing mechanisms.

Contiki offers CTC support, since X-Burst has been previously integrated into its network stack [29]. As shown in Fig. 2.4, a *virtual radio* is used to schedule CTC operations without affecting normal network traffic. The *virtual radio* allows to enqueue CTC messages and informs the application layer about incoming packets. Non-CTC related data is simply forwarded between RDC and radio layer, thus the conventional Contiki network stack remains unchainged.

2.5.2 Nexmon

Nexmon [42] is a C-based patching framework for Broadcom Wi-Fi chips. It allows to investigate, replace and modify the firmware of different Broadcom Wi-Fi radios employed in popular smartphones (e.g., Nexus 6P, Huawei P9, Samsung Galaxy S7) and the newest Raspberry Pi generations. Nexmon has been developed to exploit features of Wi-Fi chips exceeding normal operation and to analyze existing proprietary Wi-Fi firmware to allow assessment of its security. It has been previously used to build a Wi-Fi testbed for benchmarking of wireless protocols [43], a CSI extractor that allows to retrieve channel state information [25], and a reactive Wi-Fi jammer [41].

Nexmon provides the possibility to extract ROM and RAM of the existing Wi-Fi firmware to gather insights on its behavior and to extract address information, i.e., to locate funtions and variables. It offers means to add custom functionality by placing firmware patches at desired regions in the code and further allows to replace and reuse existing functions. The firmware patches can be written in C, which allows fast prototyping and makes them easily portable across different Wi-Fi chips. Example patch code, contained in Nexmon's public GitHub repository [4], allows devices to enable monitor mode, perform Wi-Fi frame injection and to exploit further features such as RSS sampling, CSI extraction or the arbitrary configuration of Wi-Fi settings (e.g., channel selection, transmission power).

2.5.3 EmberZNet PRO SDK

EmberZNet PRO is the ZigBee SDK for Silicon Labs EFR32 SoCs and can be accessed upon purchase of one of Silicon Labs' mesh networking development kits. It contains a complete ZigBee protocol stack, several examples, drivers and an *Application Framework Interface* to allow the development of custom applications. The ZigBee stack can be built and configured using *Simplicity Studio*, an Integrated Development Environment (IDE) offered by Silicon Labs. Simplicity Studio is used to define the desired ZigBee functionalities (i.e., ZigBee device type, supported ZCL clusters, employed security mechanism) of a device. It further offers a flashing tool, a console for debugging and a *Radio configurator* to adjust the EFR32 chip's physical radio settings.

2.5.4 nRF5 SDK

The nRF5 SDK [5] is a software development kit for Nordic Semiconductor's nRF51 and nRF52 Bluetooth SoCs. It is freely available and offers a variety of code examples and peripheral drivers to allow fast, easy and robust application development. Nordic provides a fully qualified BLE stack in form of the so-called *SoftDevice*. The latter is a precompiled binary file and implements BLE-related functionality, from low-level radio operations up

to application-related features. As it is a standalone binary file, it has to be flashed onto the platform along with custom code and its features can be accessed using the *SoftDevice API*.

In order to allow fast application development paired with efficiency and cross-platform compatibility, the nRF5 SDK contains several predefined Bluetooth services, such as a heart rate service, a battery service or a LED button service. Those allow standardized communication between BLE devices and are easily configurable using the *SoftDevice API*. Furthermore, Nordic offers an app called *nRF Connect*, which allows to scan for BLE devices and parses advertisement data. This way, any smartphone running *nRF Connect* can connect to available BLE devices and access their corresponding services. It can hence be used to test and control newly developed applications on nRF platforms.

Chapter 3

Related Work

This chapter describes approaches to enable communication between devices with incompatible physical layers. Nowadays, and in smart home applications in particular, multi-radio gateways are used to translate data from one protocol into another. A collection of different gateway-based solutions applied in current smart homes are presented in Sec. 3.1. Thereafter, in Sec. 3.2, an overview of existing work on cross-technology communication is given, which connects heterogenous devices without the need of additional hardware.

3.1 Building a Smart Home using Gateways

In current smart home setups, it is common to employ multi-radio gateways to allow the communication between smart home devices with incompatible communication technologies. Different types of gateways, in this context often referred to as smart home hubs, are available and explained below.

3.1.1 Vendor-specific Hubs

Most smart home manufacturers offer a smartphone app to setup and control their devices. As some devices (e.g., ZigBee or Z-Wave devices) cannot directly be controlled with the smartphone's interfaces, vendor-specific hubs are introduced (e.g., IKEA Trådfri gateway, Hive hub). A vendor-specific hub is not only used to translate and forward data, but can also act as a central control to enable home automation. However, the lack of compatibility with other smart home devices is a limiting factor regarding automated control. Furthermore, the employment of devices from different brands requires the installation and usage of separate apps, which can be a cumbersome and frustrating procedure for the user.

3.1.2 Smart Home Gateways

To tackle this problem, multi-radio smart home gateways have been developed supporting a variety of devices from different manufacturers. They thus allow to control the devices within a single app using a single gateway. These gateways are typically mains-powered devices and offer Wi-Fi to allow control using the smartphone. The setup of new smart

home devices commonly still requires the installation of each individual app [27] beforehand. In Table 3.1, several commercially available smart home gateway solutions are listed, along with the supported protocols and number of supported devices. Although most gateways are compatible with a high number of different devices already, not all of them support the three main wireless technologies in the 2.4 GHz band (BLE, ZigBee, Wi-Fi).

Vendor	**Supported Protocols**	**Supported Devices**
Samsung SmartThings	ZigBee, Z-Wave, Wi-Fi	300+
Apple Homekit	BLE, Wi-Fi	130+
Wink Hub	ZigBee, Z-Wave, BLE, Wi-Fi, Lutron Clear	120+
VeraSecure	ZigBee, Z-Wave, BLE, Wi-Fi, VeraLink	210+
Homey	ZigBee, Z-Wave, 433 MHz, Wi-Fi, BLE	70
HomeSeer	Z-Wave, Wi-Fi, Serial, Ethernet	160+

Table 3.1: Examples of commercial available smart home gateways [11].

3.1.3 Voice Assistants

In recent years, voice assistants (e.g., Alexa, Google Assistant) gained popularity and are used to control devices within the users' home. Smart speakers such as Amazon Echo and Google Home allow to control lights, plugs and other devices by voice. They support a variety of different smart home gadgets, although smart home gateways are still required as they typically employ a Wi-Fi and BLE radio only. The Amazon Echo recently got equipped with a ZigBee radio and hence allows even direct communication with ZigBee devices. The main disadvantage of voice control is, that it requires Internet connectivity during operation, as communication drivers and speech recognition are cloud-based. The latter raises also privacy concerns, as personal conversations are processed online. Furthermore, the smart home devices have to be set up conventionally prior to connection to the voice assistant and often, only a subset of the devices' features is supported.

3.1.4 Generic Gateway Approaches

The incompatibility problem has also been addressed in academic work. In [11], a generic multi-radio gateway is proposed, which automatically retrieves application and control information of the pairing device to allow communication regardless of manufacturer and technology. Other approaches aim to replace gateway devices with existing infrastrature, such as TV boxes [24] or smartphones [37].

3.1.5 Limitations

There are several disadvantages associated with gateway-based smart home solutions. The deployment of gateways introduces additional costs, as separate hardware is required. Since multi-radio gateways have to support several wireless technologies simultaneously, they are commonly powerful, mains-powered devices. The costs vary from 90€ for the

cheapest Samsung SmartThing hub without BLE support, to almost 400€ for a more powerful Homey Smart Home hub. Gateways further cause translation overhead, generate additional network traffic and lead to increasing complexity, especially, if multiple gateways are required. Finally, as gateways act as a central control, they introduce a single point of failure.

3.2 Existing Cross-Technology Communication Schemes

Cross-technology communication is an alternative way to enable data exchange between devices with heterogenous wireless technologies and does not require the deployment of additional hardware. In the following, an overview of previous work on cross-technology communication is given, divided into packet-level modulation and physical-layer modulation schemes (their difference is explained in Sec. 2.2). Thereafter, the limitations of existing CTC mechanisms and their implementation are discussed.

3.2.1 Packet Level Modulation

Esense. Esense [18], an early work in the context of cross-technology communication, is based on sensing and interpreting energy profiles and enables unidirectional data transmission from Wi-Fi to ZigBee devices. Similar to X-Burst, it defines an alphabet of packet sizes to create energy bursts of different lengths. Compared to X-Burst, it is not possible to transmit arbitrary data, but only predefined messages contained in the alphabet. It is possible to construct an alphabet of up to 100 different packets and a throughput of up to 5 kbps, in the absence of background traffic, can be achieved.

Gap Sense. In Gap Sense [53], a CTC preamble is prepended to a legacy frame to exchange information between incompatible wireless devices. The preamble is a series of signal pulses that exploits the gap between two pulses to encode data. Receivers can retrieve the information by means of energy sensing. It is a generic approach but supports only a limited set of values to be transmitted. Furthermore, it has been only showcased using a powerful SDR.

FreeBee. FreeBee [32] is communication framework that enables communication between Wi-Fi, BLE and ZigBee devices. While data exchange between Wi-Fi and ZigBee is possible in a bidirectional manner, the BLE device can only act as a receiver. The communication mechanism of FreeBee is based on shifting the timing of periodic beacon frames, known as Pulse Position Modulation (PPM). As the beacon frames are mandatory for the used wireless standards, no additional network traffic is introduced. FreeBee, however, offers only a limited data rate of 14.6 to 31.5 bps, depending on the communication direction. Furthermore, the transmission of CTC messages is only possible on BLE's advertisment channels.

B2W2: N-Way Concurrent Communication. B2W2 [19] allows N-way concurrent communication among Wi-Fi and BLE devices, i.e., a BLE device can transmit data to a Wi-Fi device while still supporting BLE to BLE and Wi-Fi to Wi-Fi communication.

Compared to the previous approaches, where timing information is used to encode data, B2W2 leverages the energy level of BLE packets to modulate data using discrete amplitude and frequency-shift keying (DAFSK). In particular, the transmission power of adjacent BLE packets is adjusted to create a sine wave. The actual data is encoded in the frequency of this wave, which can be decoded using the Channel State Information (CSI) on Wi-Fi devices. B2W2 provides unidirectional cross-technology communication only and supports a throughput of about 1.5 kbps. Although the advantages of concurrent CTC to normal communication are undeniable, B2W2 suffers from several limitations, as it has only been showcased on a powerful RF testbed and requires collisions between BLE and Wi-Fi packets during operation.

WiZig. Employing modulation techniques in both the amplitude and temporal dimension, WiZig [26] enables unidirectional communication from Wi-Fi to ZigBee devices. Its design focuses on robust data exchange in noisy environments. The data is encoded in the amplitude of packets transmitted within a specified time window. Depending on the channel conditions, which are permanently monitored, the time window is adjusted. In absence of external interference, the window can be kept short to optimize the throughput, while a long window helps to mitigate the error rate in case of a noisy channel. WiZig has been demonstrated using a SDR transmitting to a commercial ZigBee device and supports a throughput of about 150 bps in a real office environment.

3.2.2 Physical Layer Emulation

BlueBee. Exploiting physical layer emulation, BlueBee [30] enables high data rate communication from BLE to ZigBee devices. Data rates up to 225 kbps can be achieved by encapsulating a ZigBee packet within the payload of a standard-compliant BLE frame. This way, cross-technology communication is possible while being compatible with both BLE and ZigBee protocols. Furthermore, the embedded frames can be received by any ZigBee legacy device. Emulation is possible due to similar modulation techniques of BLE and ZigBee, but does not allow communication in reverse direction. BlueBee further requires an established BLE connection to allow the transmission of CTC messages.

WEBee. WEBee [34] is another CTC approach targeting ZigBee receivers. It enables Wi-Fi devices to transmit frames with a selected payload emulating a legitimate ZigBee packet. ZigBee devices can detect the ZigBee preamble embedded in the Wi-Fi frame and start the reception. The preceding Wi-Fi header and subsequent data is ignored, as it is considered as noise. WEBee allows a data rate of up to 250 kbps, the bit rate of standard ZigBee, but features unidirectional communication only.

3.2.3 Limitations

The described CTC approaches mostly lack generality regarding different wireless technologies. In particular, physical layer emulation based techniques are tailored to specific technologies by design and support only unidirectional communication. Although most packet-level modulation schemes are, in principle, applicable to different technologies and

communication directions, FreeBee is the only framework that has successfully demonstrated bidirectional communication, but cannot offer feasible data rates.

Although quite a large number of work has been published in the last years, CTC is still a rather young research topic and remains mostly confined to academia. Therefore, recent work was mostly based on prototyping, emphazising each novel technique and achievable throughput, but has not been implemented on real-world off-the-shelf smart devices. The aforementioned CTC approaches were showcased on commidity development platforms or even on powerful hardware with SDRs.

Another missing link towards the application and acceptance of CTC in a larger scale, which has not been targeted concretely yet, is the parallel operation of CTC and existing communication protocols.

In the next section, we will show that it is possible to apply CTC on real-world off-the-shelf smart home devices, paving the way for an industrial adoption of CTC.

Chapter 4

Enabling X-Burst on Wi-Fi Devices

The following chapter describes how X-Burst can be ported to a Raspberry Pi 3B+ (RPi3) and a Nexus 6P smartphone. The content is mainly based on [16], where we have ported X-Burst to the RPi3 in the context of a seminar project. Nevertheless, the concept applies to any device employing a Broadcom Wi-Fi chip: in particular, implementation details and challenges faced when porting X-Burst on the Nexus 6P, are described in more detail in Sec. 5.2.

4.1 Overview

To emphasize the technology independence of X-Burst's CTC principle, we have enabled CTC capability on Wi-Fi devices. Since Wi-Fi is among the most popular wireless technologies in the IoT domain and highly adopted, bringing CTC support to Wi-Fi platforms is a crucial step towards tackling interoperability issues.

In order to allow packet-level based cross-technology communication on Wi-Fi devices, continuous RSS sampling as well as the transmission of arbitrary packets is required. Although some platforms (e.g., TI CC3220, ESP32) provide the possibility to send custom Wi-Fi frames, they do not allow continuous RSS sampling, as it is not part of the Wi-Fi standard. As both the RPi3 and the Nexus 6P employ a Broadcom Wi-Fi chip (i.e., a `bcm43455c0` and `bcm4358`, respectively), we use Nexmon [42], a C-based patching framework, to overcome these limitations. Nexmon allows to modify and extend the firmware of Broadcom radio chips and it is thereby possible to enable frame injection as well as RSS sampling functionality.

4.2 Software Architecture

The implementation of X-Burst on the RPi3 and the Nexus 6P is twofold, as shown in Fig. 4.1. The actual CTC functionality is implemented in a user-space python script, consisting of several modules according to X-Burst's architecture explained in Sec. 2.3.3. Radio-related features are made available in the firmware. In particular, the firmware

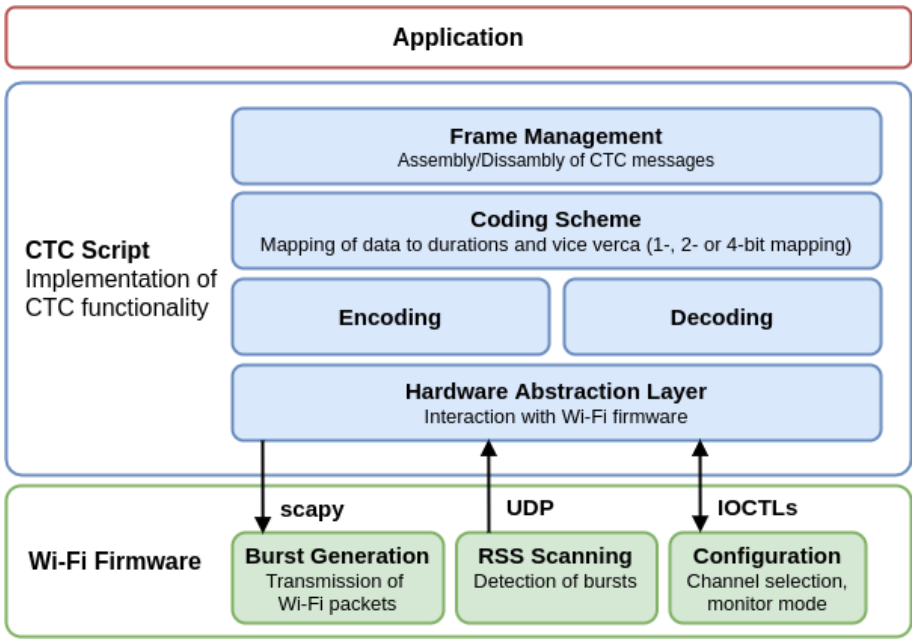


Figure 4.1: Software architecture of the Wi-Fi CTC implementation. The Wi-Fi firmware has been modified to allow burst generation and RSS sampling. The actual CTC functionality is implemented in an user-space script, which has the same format as suggested in the original Contiki implementation of X-Burst in Sec. 2.3.3. Communication between firmware and user-space is performed using the frame injection tool scapy, UDP frames and IOCTL system calls.

modifications enable frame injection, energy sensing as well as configuration of the desired Wi-Fi channel. To allow interaction between user-space application and Wi-Fi firmware, several ways of communication are available [42], which are described in more detail below.

Input/output control (IOCTL) system calls. Nexmon offers `nexutil`, a user-space program which triggers IOCTL system calls and thereby initiates synchronous data exchange with the firmware. IOCTL commands contain an identification number and a pointer to a buffer including its length. It is possible to define custom IOCTL commands and implement their corresponding function in firmware. In our implementation, IOCTL system calls are used to configure the required channel, disable the chips' power management and to enable/disable the continuous RSS sampling.

Scapy. Scapy [9] is an open-source packet manipulation program and allows frame injection of custom Wi-Fi packets. We use scapy to initiate the transmission of Wi-Fi packets, as its execution is way faster compared to IOCTL commands.

User datagram protocol (UDP) frames. To transfer data from the firmware to a user-space application, data can be encapsulated in UDP frames. If sent to the broadcast Internet Protocol (IP) address (i.e., 255.255.255.255), the frames are automatically accepted by the linux kernel and passed on to user-space applications. This approach, in contrast to the former communication mechanisms, allows data transfer without root privileges. We use UDP frames to retrieve RSS information from the firmware.

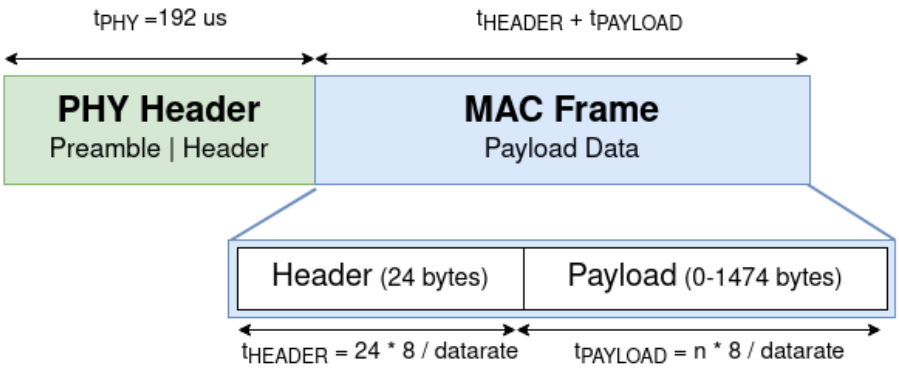


Figure 4.2: A Wi-Fi frame consists of a PHY-header and a MAC frame. The PHY-header has a fixed length of 192μs. The length of the MAC frame depends on the chosen payload size and the data rate.

4.3 Transmission

Packet Injection using JamLab-NG

To create energy bursts of different length we transmit legitimate Wi-Fi frames with varying payload size and data rate. The transmission of such bursts is based on JamLab-NG's tool `Jelly` [43], which allows to send arbitrary Wi-Fi packets. It exploits the monitor mode of Broadcom Wi-Fi chips in combination with the frame injection program scapy. Monitor mode is required to enable the transmission of Wi-Fi packets without the need of a prior connection to an access point. In monitor mode, raw Wi-Fi frames can then be forwarded to the firmware using the `L2Socket` feature of scapy. In the firmware, the transmission is initiated by calling Nexmon's `sendframe()` function. Using scapy, it is further possible to prepend a radiotap header to the Wi-Fi frame. A radiotap header [7], standardized for IEEE 802.11 frame injection, contains radio-specific parameters such as data rate and transmission power.

Minimum and Maximum Burst Length

In order to compute a feasible alphabet as described in Sec. 2.3.2, we need to determine the minimum and maximum burst length that can be generated using Wi-Fi frames. As we want to send only standard-compliant packets, the standardized Wi-Fi frame structure as illustrated in Fig. 4.2 has to be considered. A Wi-Fi frame consists of a preamble and a header, part of the IEEE 802.11 PHY layer, and a Wi-Fi MAC frame [10]. The preamble, necessary for synchronization, and the PHY header, containing information about packet format and data rate, have a fixed length of $t_{PHY} = 192\ \mu s$[1]. The Wi-Fi MAC frame contains a header of 24 byte and a variable payload. It is possible to send a maximum of 1514 bytes to the firmware. As 40 bytes are reserved for radiotap and Wi-Fi header, up to 1474 bytes of payload can be transmitted. The length of the MAC frame depends on the chosen payload size as well as the selected data rate. Hence, the duration of one Wi-Fi

[1]The IEEE 802.11 standard specifies also a *short preamble* with a length of 96 μs. However, this feature is not supported in our current implementation.

packet in microseconds is given by:

$$t_{WiFi}(n) = t_{PHY} + t_{HDR} + t_{PL} = 192 + \frac{24 \cdot 8}{DR} + \frac{n \cdot 8}{DR} \tag{4.1}$$

where DR is the data rate in Mbps and n the number of payload bytes.

Using Eq. 4.1, we can derive the minimum and maximum packet length for each supported data rate (Table 4.1) and further determine the overall minimal/maximal burst duration for a Wi-Fi device with $t_{WiFi,min} = 210\ \mu s$ and $t_{WiFi,max} = 11952\ \mu s$.

Rate (Mbps)	$\mathbf{t_{WiFi,min}}$ **(μs)**	$\mathbf{t_{WiFi,max}}$ **(μs)**
1	384	11952
2	288	6072
5.5	227	2330
11	210	1261

Table 4.1: Minimum and maximum duration of IEEE 802.11b frames.

4.4 Reception

RSS Sampling using Hardware Timer

As explained in Sec. 2.3, the reception of CTC messages is based on the detection of energy bursts by means of RSS sampling. In order to allow the continuous measurement of the RSS value on a given channel, modifications in the firmware are required. We make use of a periodic timer to trigger an existing energy detection function.

Reporting every single RSS value back to the user-space application for further processing introduces significant and uncontrollable delays due to the transmission of UDP frames through the device's network stack. Hence, the analysis of the retrieved RSS value is implemented in the firmware directly, to keep the number of transmitted UDP frames at a minimum. After each RSS measurement, the obtained value is compared against a threshold to detect the beginning and end of an energy burst. The extracted duration is returned to the CTC application in user-space.

The threshold is a fixed value stored in the firmware. It has been determined once by retrieving the noise floor and defining a threshold just above it. It is a rather aggressive threshold selection to maximize the communication range, but sensitive to external interference as any signal above the noise floor is detected. The implementation of an adaptive threshold selection would be beneficial to find a trade-off between sensitivity and robustness, but is outside the scope of this thesis and may be addressed in future work.

Limited Sampling Rate

Even though implementing the burst detection in the firmware directly helps to speed up the RSS sampling process significantly, the sampling interval is still large, compared to

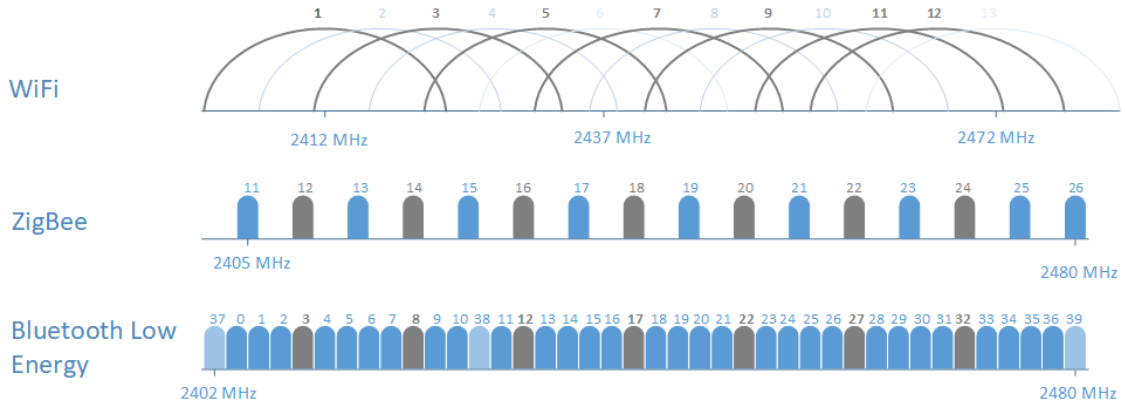


Figure 4.3: Standardized channels of Wi-Fi, ZigBee and BLE. Proposals for appropriate CTC channel selection are colored in grey. The advertising channels of BLE are colored in light blue and should not be used for CTC communication.

other devices. On the RPi3, the sampling interval amounts to $t_{sampling} = 97\ \mu s$, while devices such as the TI CC2650 and Firefly, which have been used with X-Burst previously, feature sampling intervals of approximately 23 μs and 8 μs, respectively [29]. Consequently, the limited sampling rate has to be accounted for in the alphabet computation, as the ability to distinguish two distinct energy bursts diminishes with increasing sampling interval.

4.5 General Considerations

4.5.1 Channel Selection

As described in Sec. 2.3.2, participating CTC devices have to operate in the same frequency band and agree on a common channel. Wi-Fi, ZigBee and BLE devices feature different channel configurations in the 2.4 GHz ISM band. The Wi-Fi standard specifies 13 different channels with a bandwidth of 22 MHz each. BLE and ZigBee radios, instead, employ a bandwidth of 2 MHz on 40 and 16 channels, respectively. To achieve a robust communication, the channels should be selected, such that:

- BLE and ZigBee channels overlap completely;
- BLE and ZigBee channels are located in the center of the Wi-Fi channel.

Fig. 4.3 shows the channels of Wi-Fi, ZigBee and BLE as well as proposals for appropriate channel selection.

4.5.2 Alphabet Computation

In order to find a common alphabet, we have to identify the constraints of all involved devices. As the goal is to allow communication between the three most popular technologies in the 2.4 GHz ISM band, namely BLE, ZigBee and Wi-Fi, we consider the properties of the RPi3 (Wi-Fi), the Firefly (ZigBee) and the TI CC2650 LaunchPad (BLE). The timing constraints of ZigBee and BLE devices have been investigated previously in [28].

Determining the Minimum and Maximum Burst Length

Table 4.2 shows the minimum and maximum achievable packet length of each technology (i.e., burst durations). It further contains the granularity, i.e., the minimum difference between burst durations which can be generated by the radio. ZigBee devices show the worst granularity (32 μs), due to the low data rate. Hence, only durations which are a multiple of 32 are valid choices for the common alphabet. Considering these aspects, the overall minimum and maximum burst lengths are:

- $d_{min} = 224\ \mu s$
- $d_{max} = 1984\ \mu s$

	Wi-Fi	**ZigBee**	**BLE**
$\mathbf{t_{min}}$ **(**μ**s)**	210	192	80
$\mathbf{t_{max}}$ **(**μ**s)**	11952	4256	2120
Granularity (μ**s)**	0.7 - 8	32	8

Table 4.2: Duration limits of Wi-Fi, ZigBee and BLE.

Computing the Minimum Difference between Burst Lengths

The minimum spacing between different burst durations, which can be distinguished by all devices, is determined by the sampling interval of the RPi3, as it has by far the lowest RSS sampling speed. Given the sampling interval of $t_{sampling} = 97\ \mu s$ the spacing between two burst lengths must be at least $s = 2 \cdot 97 = 194\ \mu s$. In [16] it could be shown that a spacing of $s = 192\ \mu s$ is sufficient to differentiate two burst lengths reliably and at the same time fulfills the granularity constraints of ZigBee devices.

Defining the Alphabet

Considering the timing constraints explained above, the set of durations which can be generated and correctly detected by all participating devices can be computed. Using the formula $d_i = d_{i-1} + s_{max}$ from Sec. 2.3.2 with $d_0 = 224\ \mu s$ and $s_{max} = 192\ \mu s$, the possible durations can be determined and are shown in Table 4.3.
Since $d_{max} = 1984\ \mu s$ leaves only durations $d_0 \dots d_9$ as valid choices, 4-bit mapping is not supported as it would require a set of 16 distinct durations. Hence, the selected alphabet is a 2-bit mapping consisting of durations $d_0 \dots d_3$. Such a configuration, with an alphabet size of $2^2 = 4$, proves to be the most robust configuration anyway [29].
To account for uncertainties in the detection of burst durations, Table 4.3 further includes upper and lower limits for each duration, i.e., a duration is considered valid, if it lies between this $d_{i,min}$ and $d_{i,max}$. These limits are computed with $d_i \pm \frac{s_{max}}{2}$.

i	$\mathbf{d_i}$ (μs)	$\mathbf{d_{i,min}}$ (μs)	$\mathbf{d_{i,max}}$ (μs)
0	224	128	320
1	416	320	512
2	608	512	704
3	800	704	896
4	992	896	1088
5	1184	1088	1280
6	1376	1280	1472
7	1568	1472	1664
8	1760	1664	1856
9	1952	1856	2048
10	2144	2048	2240
11	2336	2240	2432
12	2528	2432	2624
13	2720	2624	2816
14	2912	2816	3008
15	3104	3008	3200

Table 4.3: Set of duration candidates for a common alphabet.

Chapter 5

Design and Implementation

This chapter contains design considerations and implementation details of each smart home component addressed in this thesis. In Sec. 5.1 an overview of the main implementation goal is given and the corresponding requirements are identified. Furthermore, the implementation of X-Burst on the Nexus 6P smartphone (Sec. 5.2), the Trådfri light bulb (Sec. 5.3), and the Danalock V3 door lock (Sec. 5.4) is explained in more detail. Finally, we describe the final demonstration in Sec. 5.5.

5.1 Overview

The main goal of this thesis is to allow a seamless and bidirectional communication between incompatible smart home devices without the deployment of cost-intensive gateways. Towards this goal, we enable CTC functionality on three off-the-shelf platforms and build a smart home scenario as illustrated in Fig. 5.1.

The main setup consists of:

- Smartphone Nexus 6P (Wi-Fi[1]);
- Danalock V3 (BLE);
- IKEA Trådfri light bulb (ZigBee).

Using this architecture, a single application on the smartphone can be used to control all participating devices regardless of the underlying technology. Furthermore, broadcast communication is possible, i.e., any device can transmit cross-technology frames to multiple heterogeneous devices simultaneously.

Additionally, we include in the demonstration setup:

- an original IKEA Trådfri Remote (ZigBee);
- any smartphone supporting the nRF Connect App (BLE).

This allows us to showcase that CTC functionality can be implemented in a seamless way, i.e., without breaking the native communication mechanism.

[1]The Nexus 6P employs also a BLE radio. In this thesis, we use it as Wi-Fi device only.

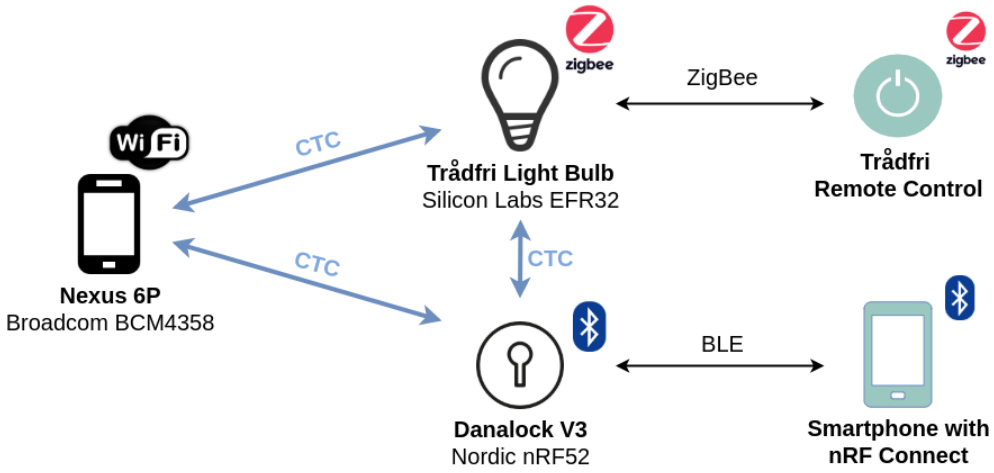


Figure 5.1: Overview of the main demonstration setup. It contains three devices employing incompatible technologies: a Nexus 6P smartphone (Wi-Fi), a Trådfri light bulb (ZigBee) and a Danalock V3 (BLE). The implementation of CTC allows a seamless communication between all of them without the deployment of multi-radio gateways. A Trådfri remote and a smartphone using the nRF Connect app are used to show the coexistence of CTC and the native communication technologies ZigBee and BLE, respectively.

Requirements. While adhering these goals, we put emphasis on three key requirements.

Adequate throughput and reliability. The CTC implementation should provide adequate throughput to allow a data exchange feasible for smart home applications. Furthermore, the reliable decoding of CTC messages, in case of external interference, must be ensured.

Minimal memory footprint. As many smart home devices are based on resource-constraint radio platforms, the memory footprint should be kept at a minimum. This way, a firmware update to introduce CTC functionality is possible for legacy devices with small memory budget.

Coexistence with native communication stacks. The CTC implementation should be integrated seamless next to the existing communication stack without breaking its functionality. In particular, the packet reception rate of the native communication mechanism (i.e., ZigBee or BLE) should not drop below 95%.

5.2 Bringing X-Burst on the Nexus 6P

The smartphone is an important and popular device in the smart home domain. Most manufacturers provide a smartphone app to control their devices. It can thus combine the functionality of several remotes in one device. While current smartphones employ both a BLE and Wi-Fi radio, it is not possible to control ZigBee devices, such as smart light bulbs, directly without the deployment of a gateway, as the underlying communication technologies are incompatible. Moreover, even though immediate communication with

BLE or Wi-Fi devices is possible, each smart home component usually relies on a separate app, since commonly different application protocols are used. These circumstances are cumbersome for the user during both setup and usage of a smart home devices. Hence, adopting X-Burst on a smartphone allows to reduce installation complexity and increases user-friendliness.

The Nexus 6P smartphone employs a `bcm4358` Broadcom Wi-Fi chip and is on the list of Nexmon's supported devices. Thus, we have chosen the Nexus 6P as this allows us to reuse existing work based on the RPi3 to enable CTC functionality.

5.2.1 Design Challenges

Detection of required Functions using ROM Extraction

Although the `bcm4355c0` employed on the RPi3 and the `bcm4358` of the Nexus 6P offer the same features, not all of them are accessible on the Nexus 6P. In particular, functions for RSS sampling and the initiation of a hardware timer have not yet been extracted. Nexmon offers the possibility to retrieve the content of the chips' ROM and RAM. Combining both yields a complete binary of the firmware, which can be analyzed to extract the required functions. Using software reverse engineering tools, such as Ghidra [1], the firmware binary can be disassembled into C-like code, which helps to understand the control flow. By comparison with existing code and other binaries, the addresses of the desired functions can be found and later be used in the firmware patch.

Alphabet Computation

$\mathbf{i}$	$\mathbf{d_i}$ (μs)	$\mathbf{d_{i,min}}$ (μs)	$\mathbf{d_{i,max}}$ (μs)
0	224	48	400
1	576	400	752
2	928	752	1104
3	1280	1104	1456

Table 5.1: Alphabet for Nexus 6P including lower and upper limits.

The alphabet can be determined in accordance with the computation considerations for the RPi3, explained in Sec. 4.5.2. The only but major difference is the limited RSS sampling rate of the Nexus 6P. While on the RPi3, a single RSS measurement can be retrieved every 97 μs, the Nexus 6P shows a sampling interval $t_{sampling} = 170$ μs, which is almost twice as long. Hence, the minimum spacing between two bursts (s) has to be adapted accordingly. Considering the Nyquist criteria, a minimum spacing of $s = 2 \cdot t_{sampling} = 340$ μs is required. Taking into account, that ZigBee devices can only create bursts which are a multiple of 32 μs, the selected spacing amounts to $s_{max} = 352$ μs and an alphabet as shown in Table 5.1.

5.2.2 Software Implementation

The software implementation resembles the architecture on the RPi3 explained in Sec. 4.2. As shown in Fig. 5.2, it consists of the modified Wi-Fi firmware, a CTC script and an

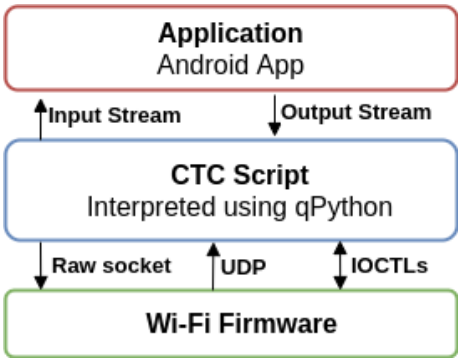


Figure 5.2: The CTC implementation on the Nexus 6P consists of three software components: the modified Wi-Fi firmware, a qPython script implementing the CTC functionality and an Android app, serving as user interface.

application.

Once the required functions of the firmware are extracted as detailed in Sec. 5.2.1, the Wi-Fi firmware can be adapted accordingly to allow RSS sampling and frame injection.

The CTC python scripts implemented on the RPi3 can be reused with minor adaptions in the HAL. As Python cannot be interpreted on smartphones natively, we make use of the qPython [6], a script engine which allows to run Python programs in Android. qPython offers no support for scapy. Therefore, the Wi-Fi frames are forwarded to the firmware using `raw sockets` from the Python's standard library. The remaining interaction between firmware and CTC script using UDP and IOCTLs works just as on the RPi3.

On top of the CTC implementation, an Android app serves as a graphical user interface. When starting the app, an additional process containing the CTC python script is created. The separation is necessary, since root privileges are required for the execution of low-level commands (IOCTLs) and the use of raw sockets. As it is not possible to request root access for these operations directly within the app, the CTC functionality is outsourced to the separate script, which can then be executed in a privileged process using Java's `ProcessBuilder` class. The `ProcessBuilder` is further responsible for the data exchange between CTC script and Android app. The communication is based on pipes, i.e., the data is forwarded from the app's standard output to the script's standard input (and vice versa) and can be accessed using Java's `InputStream` and `OutputStream`, respectively.

5.2.3 Limitations

Unstable Frame Injection

The transmission mechanism in X-Burst is based on the generation of consecutive energy bursts. The gap between those bursts should be of a certain length. If the pause is very long, the time-on-air is increased and it is more likely that external interference disturb the communication. A very short gap, on the other hand, can lead to burst detection errors, i.e., a receiver might, due to the limited RSS sampling rate, merge two bursts and decode them as one. The gap between two bursts is defined by the *radio preparation time*. The radio preparation time defines how much time the radio requires between the transmission of two packets. For the TI CC2650 and the Firefly, the preparation

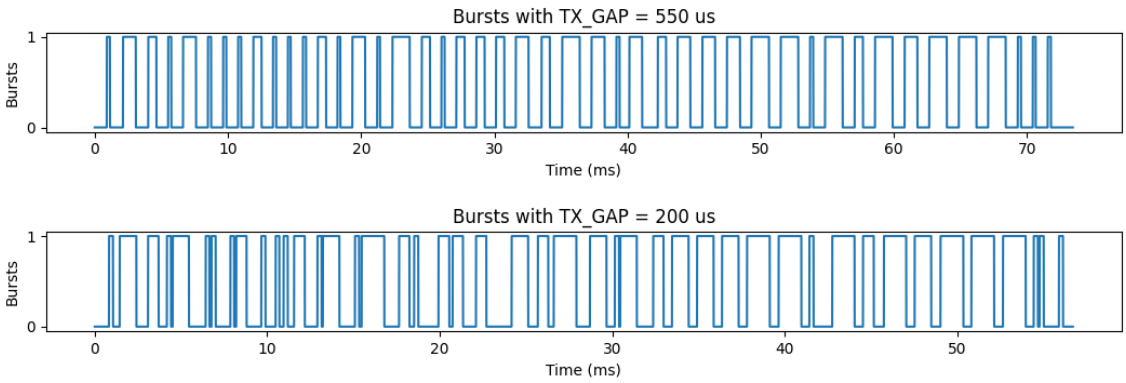


Figure 5.3: The timely distribution of energy bursts for different values of `TX_GAP`. If the transmission of two successive bursts is delayed sufficiently (`TX_GAP` = 550 μs), the bursts are generated with constant gaps in between. If the packets are queued very fast (`TX_GAP` = 200 μs), the gap between two bursts is not predictable.

time is constant with 400 μs and 240 μs, respectively. Unfortunately, on the Nexus 6P, the gap between two bursts is varying, due to uncontrollable delays introduced by the firmware. To keep the timely bursts generation as stable as possible, it is crucial to delay the transmission of subsequent bursts sufficiently, i.e., to wait a specific time (`TX_GAP`) before initiating the transmission of the next packet. Fig. 5.3 shows generated bursts recorded with a PicoScope with different selections of `TX_GAP`. If the delay between two burst transmissions is long enough (e.g., `TX_GAP` = 550 μs), an equidistant generation is achieved. With decreasing `TX_GAP`, however, the bursts tend to be transmitted in an arbitrary manner and the distinction on receiver side is aggravated.

Another problem regarding frame injection is a bug in the existing Wi-Fi chip's firmware. The repeated and fast queuing of packets can lead to a null-pointer access, which crashes the radio. Identical behavior has been encountered and solved on the RPi3. Applying the same solution on the Nexus 6P disables the frame injection mechanism and, hence, does not solve the problem. If a crash occurs, the CTC application is inoperable for a few seconds, as the firmware needs to be reloaded and reconfigured. Although the problem could not be solved entirely, a careful selection of the `TX_GAP` helps to avoid a crash.

Moreover, it appears that the operation of Bluetooth during CTC transmission has a negative impact on the burst timing. Not only does it impair the constant gap between two bursts, but crashes are provoked more frequently. It is therefore recommended to disable the smartphone's Bluetooth functionality.

Increased Sensitivity to Interference due to large Bandwidth

Wi-Fi channels have a large bandwidth of 22 MHz, compared to BLE's and ZigBee's 2 MHz channels. Hence, Wi-Fi devices are more sensitive to external interference, as the RSS sampling is performed over a wider spectrum. Further, the large bandwidth has a negative impact on the sensing range, since the narrow bursts of BLE and ZigBee devices

contribute to the energy level of the channel only in a small part. Finding a way to divide a Wi-Fi channel into smaller sub-bands would not only help to mitigate these problems, but the interference on other on-going traffic could be kept at a minimum.

Root Access during Operation

In the current implementation, root access is not only required once to install the modified Wi-Fi firmware, but continuously during operation. To transfer data from user-space into the firmware (e.g., for frame injection) high privilege functions, such as IOCTL system calls and raw sockets, are necessary. On the other hand, reception is possible without root privileges, as the RSS information is reported back to user-land using UDP frames. On the Nexus 5, it could be shown that bidirectional UDP communication is possible and would allow the operation of CTC without root privileges [4]. This approach, however, has not yet been implemented on the Nexus 6P and is outside the scope of this thesis.

5.3 Bringing X-Burst on the IKEA Trådfri Light Bulb

The IKEA Trådfri product family contains popular and cheap smart home devices, such as smart lightning, plugs, or blinds. The system is based on ZLL and allows to control the appliances wireless with a Trådfri remote or the Trådfri app.

The Trådfri GU10 light bulb has been disassembled previously and reveals an EFR32MG1, one of Silicon Labs' Wireless Gecko SoCs, integrating an ARM Cortex M4 MCU and a ZigBee radio [47]. As it is possible to replace the firmware with custom code, the Trådfri is a perfect candidate to show CTC functionality on a real-world smart home device.

5.3.1 Design Challenges

Teardown

In order to allow the flashing of custom firmware on the Trådfri bulb, hardware access to the device's circuit board is required. Therefore the bulb has to be cut open and disassembled as shown in Fig. 5.4. The Trådfri consists of a housing, a LED module and a PCB, which further includes a power supply and the small Trådfri module. The bulb's Trådfri module contains an EFR32 ZigBee SoC, a flash memory and a 38.4 MHz crystal [47]. Thankfully, several pins are exposed to the outside of the module and appropriate wires and connectors can be soldered easily. The pinout is shown in Fig. 5.5.

The chip can be programmed using the Serial Wire Debug (SWD) interface on pins `PF0-PF2` and a JTAG/SWD programmer accordingly (e.g., a J-Link EDU Base Programmer from Segger). The bulb can be turned on and off by controlling the LED modules connected to `PB12` and `PB13`. Using a PWM (Pulse Width Modulation) on these pins further allows to adjust the light's brightness. The remaining pins are used for debug purposes and a serial interface.

Using the Radio Configurator to improve CTC Performance

The IEEE 802.15.4 standard, on which the ZigBee protocol is based on, specifies that each sampled RSS value is the average over the last eight symbols (i.e., over the last 128

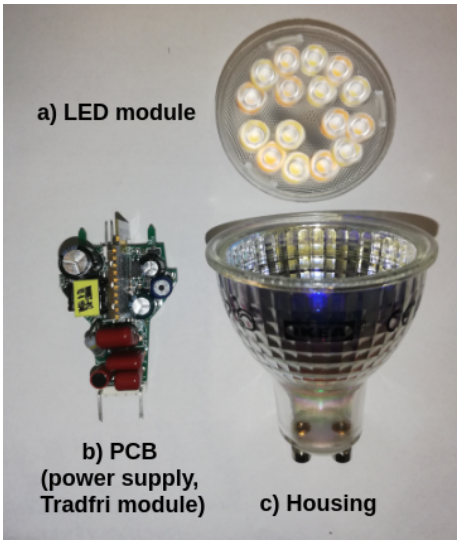


Figure 5.4: Opened Trådfri light bulb, consisting of a LED module (a), a circuit board (b) and a housing (c).

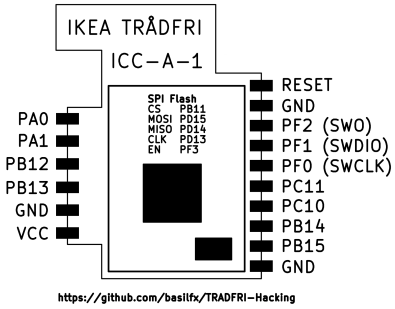


Figure 5.5: Pinout of the Trådfri module.

μs). While comparable devices with IEEE 802.15.4 radios, such as the TI CC2650 or the Firefly, allow to retrieve the averaged RSS value every 10-25 μs, on the EFR32 it is only possible to measure the RSS after $t_{Sampling} = 128\ \mu s$, resulting in a poorer decoding and reception performance.

The EFR32 platform reveals limitations also from a transmitter's perspective. Silicon Labs' Radio Abstraction Interface Layer (RAIL), which is used to access the chips' radio functionalities, allows the transmission of ZigBee packets only if the payload length exceeds at least 4 bytes. This gives a minimum burst duration of $d_{min} = 320\ \mu s$ and would lead to an alphabet containing long durations, such that the achievable throughput is limited.

To bypass these radio constraints, it is possible to change the radio configuration of the EFR32. In particular, Silicon Labs offers a *Radio Configurator* to adjust the physical radio properties as desired. We can therefore decrease the RSS update period to $t_{Sampling} = 8\ \mu s$ improve reception and reconfigure the transmission rate to allow the generation of bursts with $d_{min} = 96\ \mu s$.

5.3.2 Software Implementation

X-Burst should be implemented on the Trådfri in a way, such that the existing functionality is not compromised, i.e., the bulb can still be controlled using the original IKEA Trådfri remote. The Trådri remote is a ZLL device, which can be paired with up to ten light sources. We therefore make use of the Silicon Labs EmberZNet PRO ZigBee Stack Software, a ZigBee implementation for EFR32 devices [13], which is explained in more detail in Sec. 2.5.3. It can be configured in Simplicity Studio, Silicon Labs' own IDE, and offers a so-called application framework interface on top of the stack to implement custom applications.

To allow a seamless interaction with the Trådfri remote, the bulb is configured as a ZLL target. It enables ZigBee network formation and pairing using *Touchlink* commissioning, as described in Sec. 2.1.1. The Trådfri remote offers a button to initiate the *Touchlink* commissioning process and to connect to the bulb. We further configure the supported ZigBee clusters and implement their behavior. In particular, the Trådfri bulb has to support the `On/Off` and `Level Control` clusters and control the LED module accordingly.

To implement user-specific functionality after reception of ZigBee messages (e.g., an 'On' command from the remote), an application framework is provided. It sits on top of the EmberZNet ZigBee stack and allows to handle attribute changes of the predefined clusters. The framework further allows the execution of custom code in an event based manner. Custom events, such as timer or button events, have to be configured at compile time and are scheduled alternating with the EmberZNet ZigBee stack. In these events, it is also possible to access the device's radio functionality using the Radio Abstraction Interface Layer (RAIL).

We therefore make use of a periodic timer event to integrate X-Burst and to enable the simultaneous support of CTC and ZigBee communication, as described in more detail in the following section. The CTC functionality itself is implemented according to X-Bursts' architecture described in Sec. 2.3.3.

5.3.3 Coexistence with the existing ZigBee Stack

The access to the EFR32's radio has to be coordinated, as it is a shared resource between the ZigBee and the CTC stack. The ZigBee stack and user events, in which the CTC functionality is implemented, are executed alternating in a non-preemptive way, i.e., once a function runs, it cannot be interrupted until it returns. Furthermore, user events have full access to the radio during execution. Hence, they should be kept as short as possible to avoid negative impact on the ZigBee communication.

The basic principle of the coexistence mechanism is illustrated in Fig. 5.6. The CTC functionality is enabled periodically using a timer event with a period of $t_{Interval}$. At the beginning of each CTC 'slot', the radio is configured accordingly, i.e., the desired channel for CTC communication is selected and the RSS sampling rate is increased. Consequently, the default configuration for ZigBee communication has to be restored upon return.

In each slot, the device performs RSS sampling to sense for ongoing CTC transmissions. If no energy bursts could be detected within t_{Sense}, the next CTC slot is scheduled and the device allows the ZigBee stack to run again. This way, if no CTC communication takes place, the time allocated to the radio is kept to a bare minimum. If a burst of valid length is detected, the device stays in CTC mode, scans for a preamble, and receives the CTC message. If the message is considered valid, an acknowledgment frame (ACK) is sent immediately to inform the transmitter of the successful reception. The transmitting device, which has to send the same CTC message consecutively for $t_{Interval}$, can then stop the transmission to avoid unnecessary traffic on the channel. The ACK is a predefined set of consecutive burst durations and can be configured arbitrarily. In case of a broadcast CTC message, no ACK should be sent, as it could prevent other CTC devices to receive the message successfully.

The following timing constraints have to be considered, in order to allow a reliable CTC without affecting the simultaneous ZigBee communication:

- $t_{Interval}$ is the time between the beginning of two CTC slots. It should not be too short, as a frequent reconfiguration of the radio affects ZigBee's performance. Furthermore, the maximum CTC message length that can be reliably detected depends on $t_{Interval}$. A small value would allow only the transmission of short CTC messages. On the other hand, the interval should not be too long to avoid long delays at message reception and increased channel occupancy due to the required successive transmission.

- t_{Ack} is the time required to transmit an ACK frame and has to be determined after defining the ACK frame. Any transmitter has to wait for $t_{Ack,max}$ between the transmission of two successive CTC messages to be able to receive an ACK accordingly. $t_{Ack,max}$ is the maximum t_{Ack} of all devices participating in the CTC communication.

- t_{Sense} is the maximum time a device tries to detect energy bursts within one CTC slot. To reliably detect any ongoing CTC transmission t_{Sense} must be greater than $max(t_{Ack,max},\ d_{max} + t_{Gap,max})$. $t_{Gap,max}$ is the maximum required time between two successive energy bursts considering all devices and d_{max} is the longest duration used in the chosen alphabet.

- t_{Scan} is the time a device scans for a preamble after an energy burst has been detected. In the worst case, the CTC receiver detects an ongoing transmission at the beginning of the message and, thus, has to wait until the entire message has been retransmitted before a correct decoding is possible. Therefore, t_{Scan} has to be greater than $t_{Msg,max} + t_{Ack,max}$. ($t_{Msg,max}$ describing the maximum CTC message length).

- t_{Decode} is the time required for decoding the CTC message and is at most $t_{Msg,max}$. To avoid starvation of the ZigBee stack, the decoding process is aborted as soon as an invalid burst is detected.

5.3.4 Limitations

Since the ZigBee and the CTC stack both utilize the same radio, truly concurrent communication is not possible, i.e., it is only possible to transmit/receive with one technology at the time. In particular, if any ZigBee-related radio activity is observed at the beginning of a CTC slot, the slot is immediately rescheduled and the radio is returned to the ZigBee stack to finish its operation. Hence, ZigBee is given priority over CTC and packets of the latter type might be missed.

As mentioned above, a CTC slot is non-preemptive, i.e., once the CTC event is started it has full access to the radio and cannot be interrupted. Accordingly, if a CTC operation has been successfully initiated (i.e., if no ZigBee activity has been detected at the beginning of the slot), it can be executed as long as required. In the meantime, ZigBee packets cannot be received.

The implications on the packet reception rate of both communication mechanisms have been evaluated in Sec. 6.2.

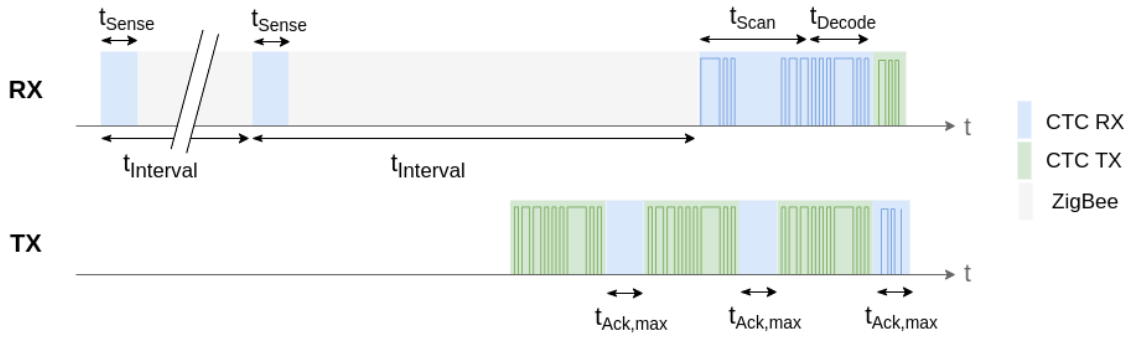


Figure 5.6: Principle of coexistence between CTC and ZigBee protocol. The device enters CTC mode periodically after $t_{Interval}$ and tries to detect ongoing transmissions. If no bursts could be extracted within t_{Sense}, the device returns the access to the radio to the ZigBee stack. Otherwise, it stays in CTC mode and receives the message. After reception, an ACK frame is sent to inform the transmitter about the successful reception. The transmitter has to send the CTC messages consecutively for $t_{Interval}$ or until an ACK is received.

5.4 Bringing X-Burst on the DanaLock Door Lock

The Danalock V3 is a BLE-enabled smart lock and employs a nRF52832 Bluetooth LE SoC. We implement X-Burst on the Danalock to complement the set of CTC-enabled real-world smart home gadgets with a BLE device.

5.4.1 Design Challenges

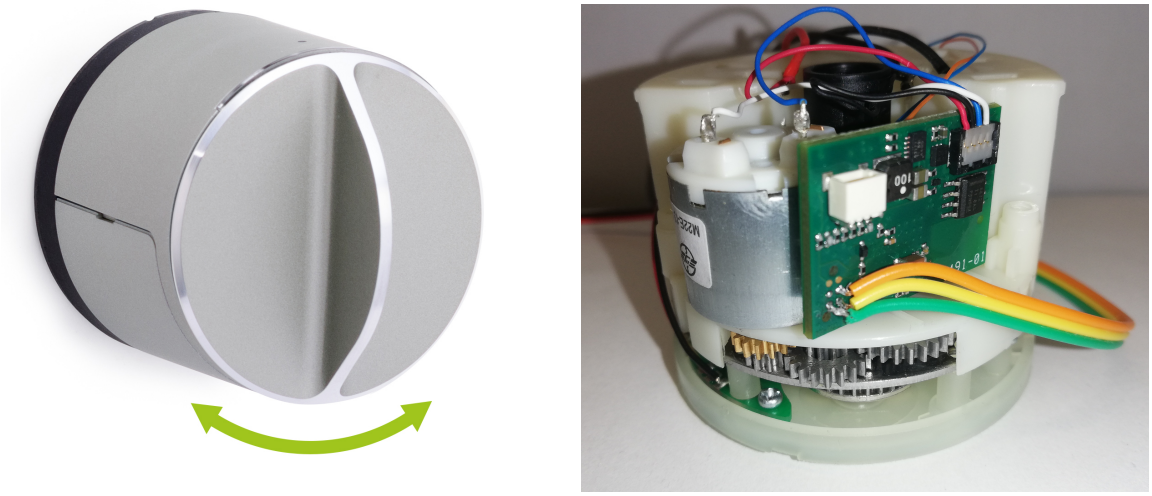
Teardown

The Danalock is encapsulated in an aluminum case and can be disassembled easily as shown in Fig. 5.7. Once screwed open, it reveals a *battery connector* (the lock is powered by four CR132A batteries providing 12V in total), a *M22E-13 DC motor* to turn the lock and a *PCB* containing the following components:

- Nordic nRF52832 Bluetooh 5.2 SoC
- TI DRV8872 DC motor driver
- TPS62177 step down converter
- RGB LED
- Button

The pinmap can be learned based on visual inspection and probing using a multimeter's continuity mode. The connections that are of our interest, i.e., the pins connected to the programming interface, motor, LEDs and button, are shown in Table 5.2.

Similar to the Trådfri, the Danalock can be programmed using the SWD interface and a Jlink Segger programmer. The SWD interface is exposed to a set of test pads on the PCB, where it is possible to attach small wires that can then be connected to the programmer device.



(a) Danalock V3 [21]

(b) Danalock V3 opened

Figure 5.7: Danalock V3 before (a) and after (b) teardown. The lock contains a DC motor, a circuit board and a battery connector.

Name	**Pin**	**Description**
Button	P0.07	Small user button, accessible to reset device
LED Red	P0.27	Red part of RGB LED
LED Green	P0.26	Green part of RGB LED
LED Blue	P0.25	Blue part of RGB LED
Motor 1	P0.22	Connected to pin1 of DC motor, used to turn motor left
Motor 2	P0.23	Connected to pin2 of DC motor, used to turn motor right
Motor Enable	P0.05	Enables motor driver, must be set to 'high' to allow DC motor to work
UART TX	P0.12	Additionally available pin, used for UART communication
UART RX	P0.11	Additionally available pin, used for UART communication
Reset	P0.21/RESET	SWD Interface, exposed to test pad
SWD Clock	SWCLK	SWD Interface, exposed to test pad
SWD Data	SWDIO	SWD Interface, exposed to test pad
Ground	GND	0V, exposed to test pad

Table 5.2: Pinmap of the Danalock V3.

Further hardware modifications are necessary to allow the use of the serial interface for debugging purposes. In particular, we add wires to available pins (`P0.11` and `P0.12`) and attach a FTDI adapter to enable UART communication. Additional user interaction is possible using the RGB LED (`P0.25-P0.27`) and a button (`P.07`).

In order to control the motor, i.e., to actually turn the lock, the TI DRV8872 motor driver must be enabled. This can be done by setting the corresponding pin (`P.05`), which controls a transistor-switch to supply the motor driver accordingly. By default, the motor driver is not powered to minimize the lock's energy consumption. The motor itself can be controlled via PWM on `P0.22` and `P0.23` to turn the lock left and right, respectively.

5.4.2 Software Implementation

The goal of the software implementation is to allow interaction with the lock using CTC while it can still be controlled with conventional BLE communication. Towards this goal, we make use of the Nordic nRF5 SDK [5]. The nRF5 SDK is a software development kit for nRF52 and nRF51 Bluetooth SoCs and contains various drivers and examples for fast application development on Nordic platforms. It further features a Bluetooth 5.1 qualified BLE protocol stack, the so called *SoftDevice*. The *SoftDevice* is a precompiled binary file which implements BLE-related features and can be flashed along with custom code to build BLE based applications.

The nRF5 SDK contains a set of predefined BLE services and corresponding examples. Our implementation is based on the `LED Button Service`, as it allows one BLE device (i.e., a central) to control the LED value of another device (i.e., the peripheral). In our case, the LED value corresponds to the status of the lock. The `LED Button Service` requires a *connection-based communication* for data exchange, as described in Sec. 2.1.2. Consequently, the Danalock's nRF52 is configured as a BLE peripheral and handles changes of the LED value accordingly, i.e., it turns the motor to open/close the lock. The smartphone, on the other hand, acts as a central device. Nordic Semiconductor offers an app called *nRF Connect*, which allows to scan for BLE devices, parses their advertisement and service data and, after connecting, allows to adjust their attributes (i.e., the LED value) accordingly. This way, any smartphone running the *nRF Connect* app can be used to control the lock using BLE.

In order to enable CTC functionality along with BLE communication, access to the radio is required. In general, when using the *SoftDevice*, the radio functionalities are not available but have to be explicitly requested using the Timeslot API, which is explained in more detail below. Similar to the code on the Trådfri, the CTC implementation itself follows X-Burst's modular design described in Sec. 2.3.3.

5.4.3 Coexistence with the existing BLE Stack

In principle, the considerations previously made for the Trådfri in Sec. 5.3.3 apply for the Danalock as well, i.e., the access to the radio must be timely coordinated between the CTC and BLE/ZigBee stack. In contrast to the Trådfri's EFR32 chip, where the radio is freely available to custom events, the nRF52's radio is only accessible through the Timeslot API.

The Timeslot API offers primitives to request the radio for a given amount of time within a specific interval. Hence, it is possible to periodically obtain a timeslot to perform

CTC communication. The access to the radio is granted if currently no BLE communication takes place and the transmission of advertising packets is not scheduled before the end of the requested timeslot. Otherwise, the timeslot is delayed to the earliest possible point in time where the BLE stack is idle. Once granted a timeslot, the radio is configured in BLE mode, which is feasible to create adequate energy bursts and perform RSS sampling. However, the radio has to be turned on at the beginning of the slot, as it is disabled by default. The application (i.e., the CTC implementation) can return the access to the radio to the *SoftDevice* at any time and is obliged to do so before the end of the timeslot to prevent a hardfault.

5.4.4 Limitations

As the *SoftDevice* can deny a timeslot request, BLE communication is given priority over CTC implicitly. Furthermore, the longest timeslot which can be requested is 100 ms and thus limits the maximum CTC message length, as detailed in the evaluation in Sec. 6.2.3. The Timeslot API offers the possibility to ask for a timeslot extension of up to 128s. Since the grant of such an extension is uncertain, it is not used for CTC reception to keep the timing properties as predictable as possible and allow an implementation according to Sec. 5.3.3. For transmission, however, the timeslot extension is leveraged to enable the transmitting device to send packets consecutively for $t_{Interval}$. This implies, that if the timeslot extension is not successful, the messages are not transmitted for the entire CTC period and might thus be missed by the receiving devices. Unsuccessful transmissions have to be handled in the application accordingly.

Contrary to the Trådfri, where the original IKEA remote can be used to control the lamp, the original Danalock app is not supported. The replication of the Danalock's pairing process and GATT services has not been possible due to its encryption mechanism. Therefore, the *nRF Connect* app is used to demonstrate the coexistence between CTC and the conventional BLE stack. Another limitation is that the communication with the *nRF Connect* app is not secured (i.e., any BLE central can control the lock). The CTC-enabled Danalock is thus feasible for demonstration purpose only.

5.5 Bringing it all Together: Demonstrator

To demonstrate the convenient control of smart home devices using cross-technology communication and to complete the demonstration setup as described above (Fig. 5.1), we designed an Android app as shown in Fig. 5.8. It allows to turn on, switch off and dimm the Trådfri light bulb and to lock/unlock the Danalock V3 door lock. The ability to transmit CTC broadcast messages can also be demonstrated by controlling both devices simultaneously. The status displays of the light bulb and the lock gives information of their current state, while the debug window shows CTC stack internal events, such as the initiation of a CTC transmission or the reception of an ACK frame.

All devices have to be configured with the same alphabet (e.g., 4-bit mapping with $d_0 = 224\ \mu s$ and $s = 92\ \mu s$), and a common preamble and acknowledgment have to be defined. The Trådfri and the Danalock are assigned a fixed 1-byte address to allow the distinction in case of unicast transmissions. They both run the CTC stack and their conventional communication stacks ZigBee and BLE, respectively, in parallel. Hence, the

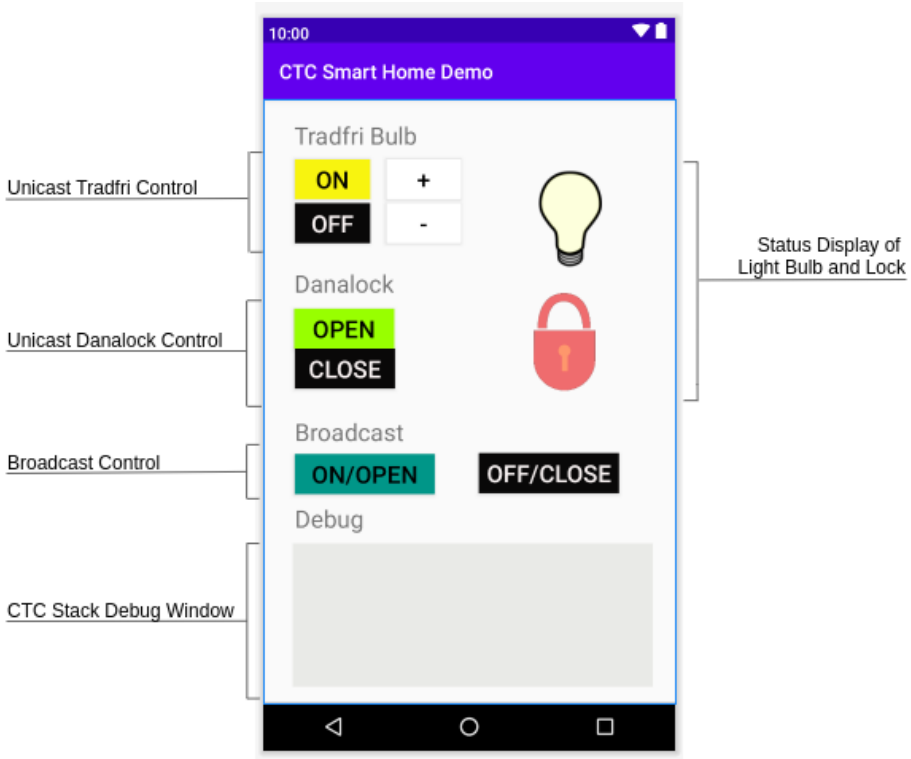


Figure 5.8: Screenshot of the smart home demonstrator app. The Trådfri light bulb and the Danalock V3 door lock can be controlled separately using unicast CTC messages, or simultaneously using broadcast communication. A debug window provides information of the smartphone's running CTC stack (e.g., encoding information, received ACKs).

light bulb can still be used with the original IKEA Trådfri remote, while the Danalock can be controlled by BLE-enabled devices.

This demonstrator shows that a gateway-free smart home setup is possible, despite incompatibilities of the employed wireless technologies. It is worthwhile noting, that the setup is not limited to the above mentioned smart home devices, but can easily be extended with further CTC-enabled platforms.

Chapter 6

Evaluation

This chapter contains the results of the experimental evaluation of the CTC implementation. In Sec. 6.1, the nominal properties of X-Burst on the new hardware platforms (i.e., the Nexus 6P, the Trådfri and the Danalock V3) are investigated, i.e., we evaluate the CTC performance only, hence the existing communication protocols are disabled. In particular, the achievable throughput and the packet reception rate in the presence of external radio interference is obtained. We further evaluate the communication range. Sec. 6.2 contains an analysis of the memory footprint and experiments tackling the coexistence mechanism. In particular, the PRR and throughput at simultaneous transmission of CTC and ZigBee/BLE packets are shown. Finally, Sec. 6.3 shows the CTC smart home demonstrator in action.

6.1 X-Burst Properties

In the following section, we evaluate the performance of X-Burst on the newly added platforms (i.e., the Nexus 6P, the Trådfri and the Danalock V3). In particular, the throughput at different payload sizes is investigated along with the packet reception rate in the presence of external radio interference. Furthermore, an evaluation of the achievable communication range is given.

6.1.1 Experimental Setup

The experiments are performed in an office environment, i.e., the devices are not completely shielded from background noise. However, an appropriate channel selection is used to minimize external interference. If not stated otherwise, the devices are placed 1 m apart and a common 2-bit alphabet, as derived in Sec. 5.2.1, is applied.

In addition to the aforementioned devices, the Raspberry Pi 3B+ (Wi-Fi), the TI CC2650 LaunchPad (BLE) and the Firefly (ZigBee) are included in the setup to give more comprehensive evaluation results and show the compatibility with existing X-Burst enabled devices. For convenience, we use a Thunderboard Sense development platform instead of the actual Trådfri bulb for evaluation. The Thunderboard Sense employs the same ZigBee SoC (EFR32) as the Trådfri, but offers more debug capabilities (e.g., buttons and GPIO pins) and a USB programming interface. Similarly, the Danalock is replaced

with a nRF52840 Development Kit as the chips of both devices belong to the same BLE SoC family.

As the focus of this section lies on the properties of the cross-technology communication mechanism, the devices are configured such that they perform CTC only, i.e., the native BLE/ZigBee functionality is disabled.

6.1.2 Nominal Throughput

As discussed in Sec. 2.3.4, the achievable throughput depends on the chosen alphabet, i.e., the set of predefined burst durations. In our implementation, as all devices have to agree on a common alphabet to allow bidirectional and seamless communication, the throughput is mainly limited due to the slow RSS sampling rate of the Nexus 6P. The throughput further depends on the chosen payload, as higher values (e.g., '0xFF') are translated into longer durations and hence require more time for transmission. The messages in the following experiments contain equally distributed byte values (i.e., '0x01 - 0xEF') to account for these considerations and allow to compute the average value of the achievable throughput.

To evaluate the throughput, we transmit 10x100 CTC messages back-to-back from one device to another. Recording the transmission time and the number of received messages, the throughput can be calculated. Each CTC message is encapsulated in a 1-byte header and a 1-byte checksum, which allows to verify if the message has been received correctly. Only the payload bytes are considered for the throughput calculation, i.e., header and checksum are omitted. Hence, if more bytes are transmitted within one message (i.e., a higher payload size is chosen), the throughput can be increased as less overhead is introduced.

Throughput of Nexus 6P in Transmission Mode

Fig. 6.1 shows the throughput of the Nexus 6P when transmitting CTC messages of different payload size to several counterparts. It can be seen that the throughput differs depending on the receiving device. As the transmission time is constant, the different results are based on the timing properties (e.g., RSS sampling rate) and hence the reception capabilities of each device. The BLE devices (TI CC2650 and Danalock) outperform the other devices, as they are capable of instantaneous RSS measurements. Similar, yet little worse results are measured on the ZigBee devices (Trådfri and Firefly). Their radios, based on IEEE 802.15.4, provide averaged RSS measurements. A significantly lower throughput can be observed on the Raspberry Pi, due to its slow RSS sample rate.

As expected, the throughput increases with higher payload sizes. At a certain payload size, however, the packet reception rate starts to decrease, leading to a decline of the throughput. This is due the increased chances of bit corruptions, and is particularly evident on the RPi3. Nevertheless, a throughput of more than 1 kbit/s for each device is possible.

Throughput of Nexus 6P in Reception Mode

Fig. 6.1 shows the throughput of the Nexus 6P when receiving CTC messages of different payload size from several counterparts. The achievable throughput for each device deviates significantly, due to differences in the radio *preparation time*, i.e., the time required between

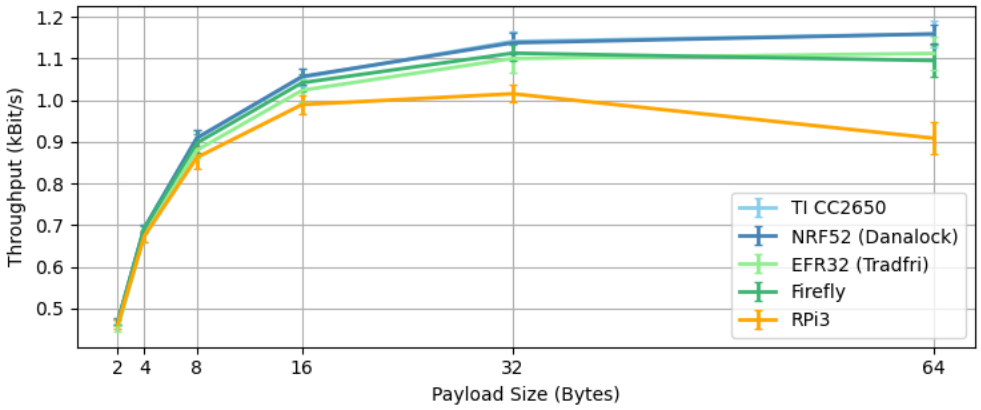


Figure 6.1: Throughput of Nexus 6P when transmitting to several counterparts depending on the payload size.

the transmission of two energy bursts. The Firefly, the Danalock and the Trådfri all feature a *preparation time* of around 200 μs and hence, allow a similar throughput. The TI CC2650 LaunchPad, with a *preparation time* of 400 μs cannot compete with this transmission speed. Again, the RPi3 shows the worst performance, as it requires a gap of around 480 μs between the transmission of two bursts. The previous experiments show, however, that bidirectional communication based on X-Burst is also possible for devices employing the same wireless technology (i.e., Wi-Fi to Wi-Fi).

In general, the throughput of the Nexus 6P in transmission mode (Fig. 6.1) is lower than in reception mode (Fig. 6.2) and is based on the long *preparation time* of the Nexus 6P (about 800 μs on average).

Summarized, the Nexus 6P can receive CTC messages with a data rate of 1.2 to 1.8 kbit/s and transmit with rates from 1 to 1.15 kbit/s, depending on the selected device. Considering the smart home devices addressed in this thesis, namely the Danalock and the Trådfri, bidirectional communication with a throughput of 1.1 to 1.7 kbit/s can be achieved.

Throughput of Danalock and Trådfri

The previous experiments (Fig. 6.1 and Fig. 6.2) have shown that the Danalock and the Trådfri can communicate with the Nexus 6P with data rates up to 1.7 kbit/s. The achievable throughput for communication between Danalock and the Trådfri has not been shown explicitly yet. However, due to their superior timing properties compared to the smartphone, it can be safely assumed that they receive each other's CTC messages at least as well as the Nexus 6P.

To show that both devices are capable of even faster communication, we apply an alphabet consisting of shorter durations and retrieve the throughput. In particular, a 4-bit mapping with $d_0 = 224$ μs and $s = 92$ μs is used and messages with 16 bytes of payload are transmitted. Fig. 6.3 shows the throughput of the Danalock and the Trådfri in reception mode. It can be seen that both devices support a throughput of more than 2.7

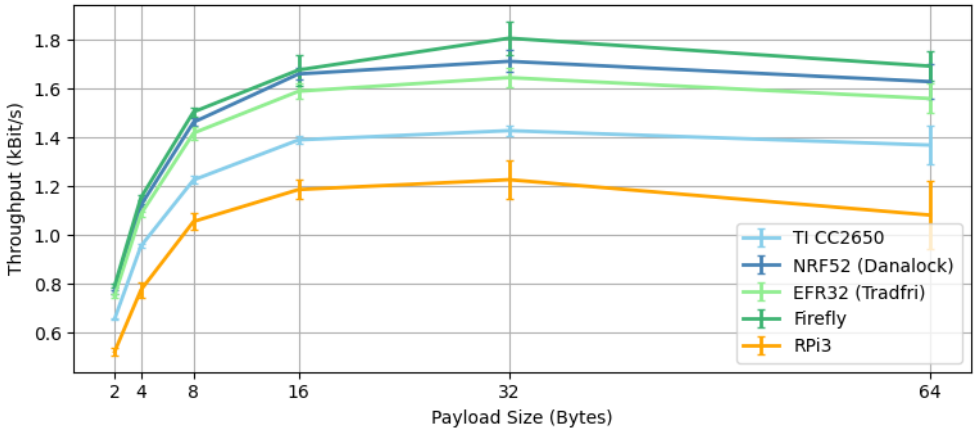


Figure 6.2: Throughput of Nexus 6P when receiving from several counterparts depending on the payload size.

kbit/s when communicating with each other. The Nexus 6P is not able to retrieve CTC messages composed of such an alphabet due to its slow RSS sampling rate. Transmission, however, is possible and a throughput of more than 1.6 kbit/s can be achieved, which is an enhancement of more than 30% compared to throughput achieved previously. The difference between the data rate of the Nexus 6P compared to the Danalock and Trådfri is again based on the different *preparation times* of the devices.

6.1.3 Packet Reception Rate in the Presence of external RF Interference

In the following, the influence of external interference on X-Burst's robustness is investigated. In particular, the PRR under three different interference scenarios is shown.

First, the PRR is retrieved in the absence of radio interference, building a reference for further measurements. Additionally, two Wi-Fi interference patterns are created: audio streaming (i.e., using Spotify) and video streaming (i.e., from YouTube). In order to allow the repeatable creation of Wi-Fi traffic, we record the desired network packets using a TP-Link USB Wi-Fi adapter. The adapter can be used in monitor mode to capture network traffic, which can be further filtered using the open-source packet analyzer Wireshark. This way, we create a set of Wi-Fi packets corresponding to the mentioned scenarios. During the experiments, they are repeatedly transmitted using the `tcpreplay` command.

To obtain the PRR, the experimental setup described previously is applied. For each measurement point, we send 10x100 16-byte long CTC messages back-to-back from the Nexus 6P to several counterparts and vice versa. The TP-Link adapter is placed at a distance of 1 m to the receiving device. The Wi-Fi interference is created with a transmission power of 13 dBm on the same Wi-Fi channel used for CTC communication.

In Fig. 6.4 the PRR obtained under different interference patterns can be seen for both transmission and reception of the Nexus 6P. As expected, the reception rates decrease

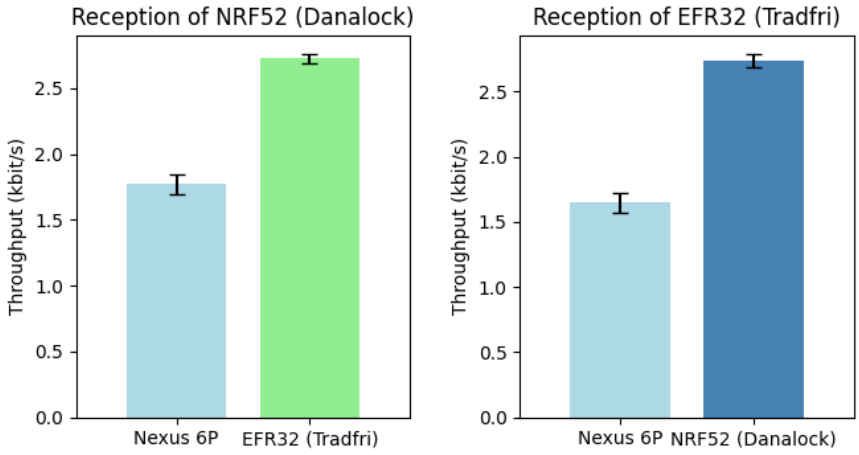


Figure 6.3: Throughput of Danalock and Trådfri in reception mode using a 4-bit mapping with $d_0 = 224\ \mu s$ and $s = 92\ \mu s$.

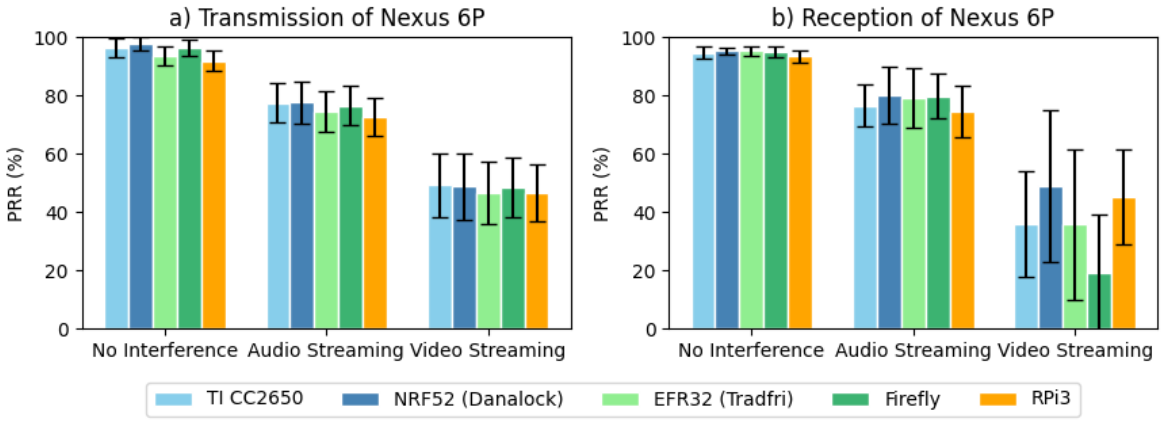


Figure 6.4: Packet reception rate at different interference scenarios. The PRR is obtained once for the transmission from Nexus 6P to several CTC enabled devices (a) and vice versa (b).

with rising severeness of the interference. In absence of any especially generated external traffic, the PRR is around 95% for each device and communication direction. The presence of Wi-Fi interference lets the PRR decline significantly. While reception rates are still adequate during audio streaming (72% to 80%), they drop below 50% at interference introduced by video streaming.

Considering the latter scenario, it can be observed that the PRR is constant for each device if the Nexus 6P acts as a transmitter. Contrary, if the Nexus 6P receives the CTC messages, the PRR varies from device to device. These observations are based on the experimental setup combined with differences in the *preparation time* of each device. The Nexus 6P sends broadcast messages with a constant *preparation time* and hence, the time required for the transmission time of one set of test messages is constant. This transmission time differs for the other devices and is shorter compared to the Nexus 6P. Thus, variations in the interference patterns (e.g., due to buffering during video streaming) affects the PRR of each test point more severely. In addition, although sending with the same transmission power, the antenna characteristics and RF front ends of the devices vary and affect the ability to decode energy bursts in presence of interference.

6.1.4 Communication Range

This section contains an evaluation of the achievable communication range between the Nexus 6P, the Danalock and the Trådfri. More precisely, the PRR at different distances between receiver and transmitter is investigated.

Again, we transmit 10x100 16-byte CTC messages to obtain the PRR. The experiment is performed in a corridor, which allows us to take measurement at distances in a range of 5 to 25 m. All devices are mounted on stands about 1 m above the ground. The PRR is obtained for each of the three devices (Nexus 6P, Danalock, Trådfri) receiving from the others. The results are shown in Fig. 6.5.

It can be seen, that both the Trådfri (b) and the Danalock (c) can receive CTC messages from all devices within a distance of 25 m while the PRR stays constant for the entire range. On the Nexus 6P smartphone (a), on the other hand, a decline of the PRR can be observed at 20 m, although a communication is still possible, and significantly worsens at a distance of 25 m. When conducting the experiment, it could be noticed that the smartphone's orientation has a substantial influence on the RSS sampling capability. It is thus kept constant during the entire measurements (i.e., in an upright position, the back facing the counterparts).

In summary, all devices feature a communication range of at least 25 m, while the Nexus 6P can receive CTC messages reliably for more than 15 m. The limited reception range of the Nexus 6P is probably based on the RSS sampling bandwidth, i.e., the Nexus 6P's Wi-Fi radio measures the energy level on the entire 22 MHz Wi-Fi band. The impact of the 2 MHz energy burst generated by the counterparts on the measured RSS value is therfore rather low. Nonetheless, considering a smart home application where many devices are located within one room, these numbers are sufficient.

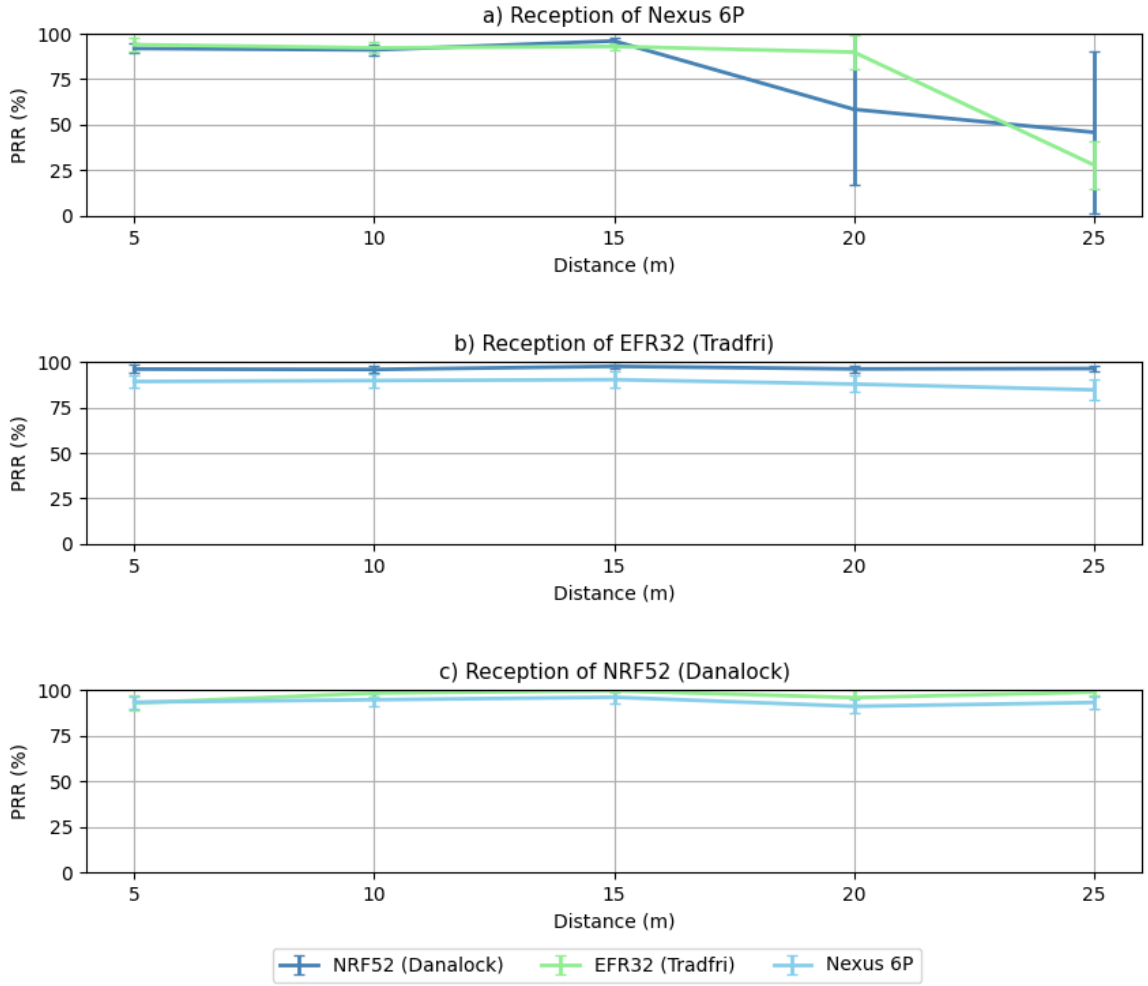


Figure 6.5: Packet Reception Rate (PRR) at different distances for Nexus 6P (a), Trådfri (b), and Danalock (c) in reception mode.

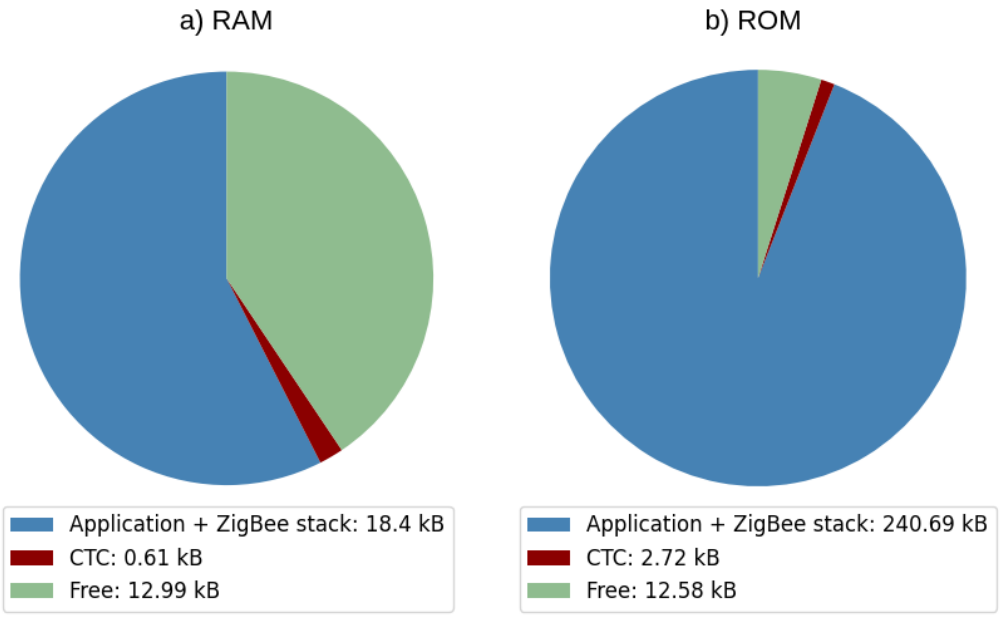


Figure 6.6: Memory footprint of the Trådfri light bulb, divided into RAM (a) and ROM (b) usage.

6.2 Coexistence with existing Communication Stacks

In this section, we evaluate the coexistence mechanism as described in Sec. 5.3.3 in order to show that the concurrent use of CTC and other existing communication technologies is possible. Furthermore, we analyze the memory footprint of the CTC stack compared to the existing protocols.

6.2.1 Memory Footprint

We analyze the memory on the Trådfri and the Danalock in terms of RAM and ROM usage. The required memory for the CTC implementation amounts to:

- 0.61 kB of RAM and 2.72 kB of ROM on the Trådfri;
- 0.67 kB of RAM and 3.16 kB of ROM on the Danalock.

Compared to the memory required for the existing communication stack (see Fig. 6.6 for the Trådfri and Fig. 6.7 for the Danalock), the CTC implementation is very lightweight and thus feasible for resource-constrained devices. The main portion of the memory consumption is caused by buffers which are required to store the extracted burst durations and the corresponding data. Limiting the maximum payload size could therefore, if required, reduce the memory footprint further.

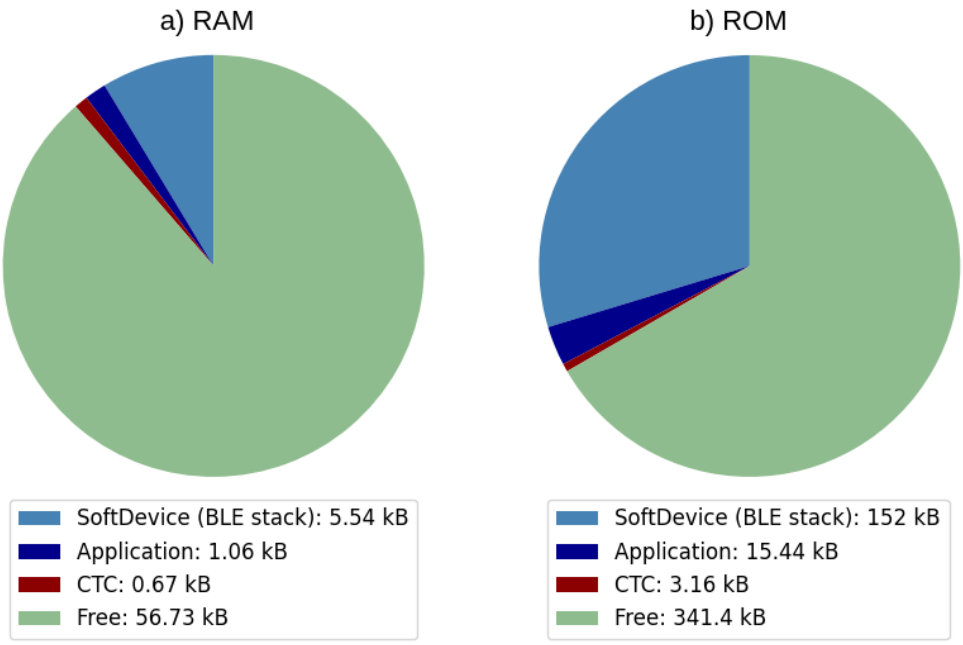


Figure 6.7: Memory footprint of the Danalock V3 door lock, divided into RAM (a) and ROM (b) usage.

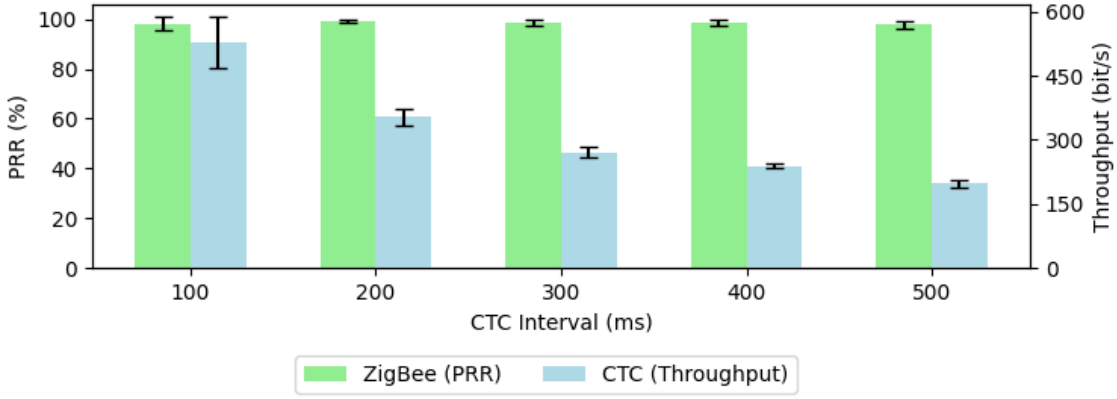


Figure 6.8: PRR of ZigBee communication and throughput of CTC for concurrent ZigBee/CTC transmission.

6.2.2 Coexistence of ZigBee and CTC

Experimental Setup

To demonstrate that a coexistence between ZigBee and CTC is possible, we concurrently transmit ZigBee and CTC packets to the Trådfri and obtain the PRR of the ZigBee communication and the throughput of CTC. Therefore, the Nexus 6P smartphone sends 10x100 16-byte unicast CTC messages consecutively, while determining the throughput. A 4-bit mapping (as used in Sec. 6.1.2 already) is applied. Simultaneously, we use a Silicon Labs Wireless Development Kit, acting as a ZigBee remote, to periodically transmit a ZigBee `on/off` command and obtain the PRR. The CTC and ZigBee communication takes place on separate radio channels.

Throughput depending on CTC Interval

First, we investigate the CTC throughput at different configurations of the CTC interval ($t_{Interval}$). ZigBee packets are transmitted with a constant period of $t_{ZigBee} = 500\ ms$. As shown in Fig. 6.8, the achievable throughput highly depends on the chosen CTC interval. As a short $t_{Interval}$ allows the device to scan for CTC messages more frequently, the throughput can be maximized if $t_{Interval}$ is decreased. Compared to the observations in Sec. 6.1.2, where the radio is used for CTC communication only, the latter is significantly lower, as only one packet per CTC interval can be received. Still, a throughput of more than 500 bit/s can be achieved.

The experiment also shows, that the existing ZigBee communication stack can still operate adequately, as a PRR of about 99% can be observed. Single ZigBee packets get lost if a CTC frame and a ZigBee message collide time wise, i.e., if a ZigBee message is transmitted during the reception of a CTC frame.

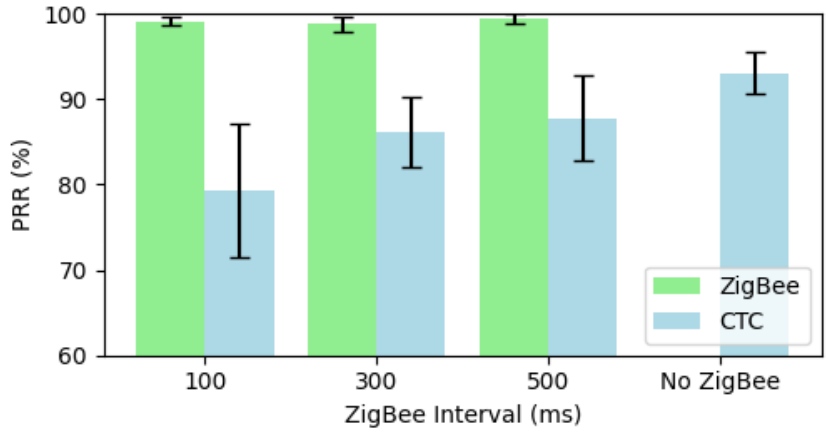


Figure 6.9: PRR of CTC and ZigBee communication for different values of the ZigBee transmission interval.

PRR depending on Traffic Load

The higher the traffic load, i.e., the more frequently messages are transmitted, the higher is the chance for collisions. Since the ZigBee remote is not bound to any schedule, but can transmit at arbitrary points in time, collisions cannot be avoided completely but have to be accounted for in higher layers (e.g., using retransmissions). Fig. 6.9 shows the PRR for both ZigBee and CTC at different traffic loads, i.e., for different intervals between two ZigBee messages at a constant $t_{Interval} = 200\ ms$. It can be seen that a high amount of frequent ZigBee messages has negative consequences on the CTC reception rate, highlighting the implicit prioritization of ZigBee traffic, as pointed out in Sec. 5.3.4. It is in particularly evident at small intervals, as pronounced with a high variance shown at $t_{Interval} = 100\ ms$ and $t_{ZigBee} = 100\ ms$ in Fig. 6.8 and Fig. 6.9, respectively.

6.2.3 Coexistence of BLE and CTC

Maximum Message Length

As mentioned in Sec. 5.4.4, the Timeslot API (used to allocate the radio for CTC operations) allows to request timeslots up to a length of $t_{Slot} = 100\ ms$ only. Considering the coexistence mechanism explained in Sec. 5.3.3, we can thereby derive a maximum message length $t_{Msg,max}$, which can be reliably received within one slot.

In the worst case, the CTC reception is started at the beginning of one message, right after the first burst and without detecting the preamble. Hence, the device has to wait until the entire message has been retransmitted before a correct decoding is possible. In this case, the reception time of one message amounts to $t_{RX} = 2 \cdot t_{Msg} + t_{Ack,max}$. With $t_{RX} = t_{Slot}$, the maximum message length is given by $t_{Msg,max} = \frac{t_{Slot} - t_{Ack,max}}{2}$.

In the current implementation, a fixed, predefined preamble with $t_{Ack,max} = 6\ ms$ is used, which gives a maximum message length of $t_{Msg,max} = 47\ ms$. The actual maximum amount of payload bytes that can be transmitted within one message depends on the

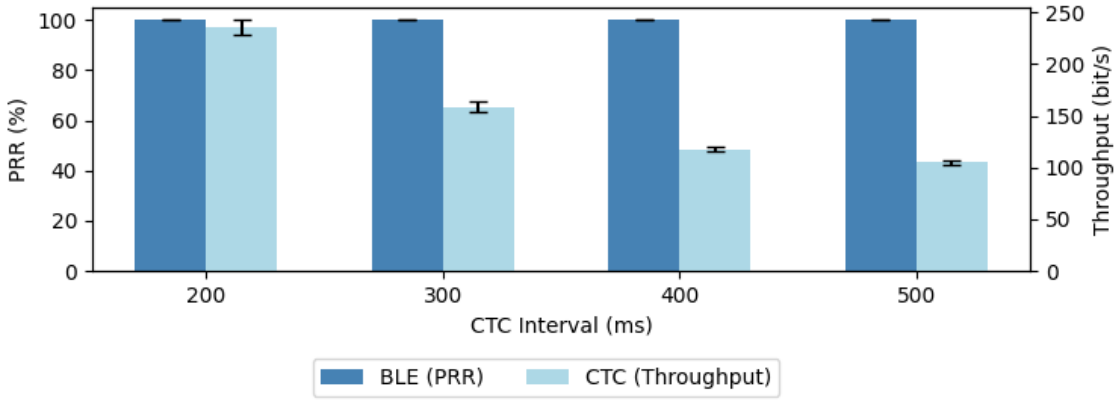


Figure 6.10: PRR of BLE communication and throughput of CTC for concurrent BLE/CTC transmission.

employed alphabet, the selected preamble, the message overhead (e.g., header, checksum), as well as the payload content. It is further platform-dependent, as the radio *preparation time* affects the transmission speed.

Experimental Setup

The experimental setup is in principle identical to the ZigBee-based coexistence evaluation. The Nexus 6P smartphone transmits consecutive unicast CTC frames to the Danalock, while another device periodically sends BLE packets. To satisfy the maximum message length retrieved before, CTC frames with a payload of maximum 7 bytes (with equally distributed values) can be transmitted. We use a nRF52840 Development Kit as BLE counterpart. It is configured as BLE Central to transmit one BLE packet per connection interval t_{BLE}.

Throughput depending on CTC Interval

Fig. 6.10 shows the CTC throughput for different CTC intervals at a constant BLE connection interval of $t_{BLE} = 500\ ms$. As expected, the throughput decreases with rising CTC interval. Compared to the previous experiment conducted on the ZigBee device, the throughput is lower due to the limited message length. A throughput of up to 235 bit/s can still be considered feasible for smart home related communication. Furthermore, the CTC interval cannot be arbitrarily low, as the *SoftDevice* is not able to schedule its operations accordingly. The minimum CTC interval that allows the CTC stack to run properly is 200 ms.

Contrary to the ZigBee communication, not a single BLE packet gets lost. Thanks to the connection-based communication scheme, the BLE packets are received with a reception rate of 100% regardless of any ongoing CTC activity.

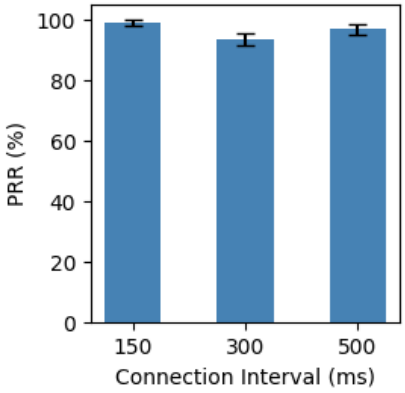


Figure 6.11: PRR of CTC at simultaneous BLE communication for different connection intervals.

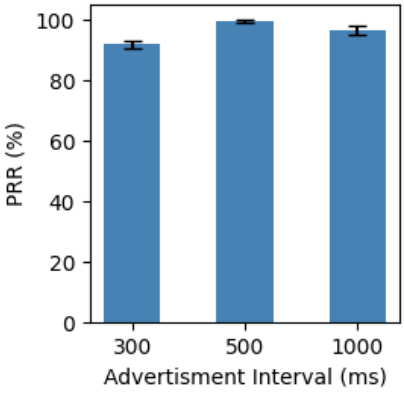


Figure 6.12: PRR of CTC in non-connected state at different advertising intervals.

PRR depending on Traffic Load

We evaluate the PRR using different traffic loads for the BLE communication. Therefore, the connection interval is varied, while the CTC interval is kept constant at 200 ms. As shown in Fig. 6.11, the PRR of the CTC communication is very high regardless of the used connection interval. This is due to the schedule mechanism of the Timeslot API combined with the short size of the CTC packets: if a CTC slot cannot be granted due to an ongoing BLE reception, the slot is rescheduled at the earliest point in time and the reception of a CTC packet can still be possible.

Even if no actual BLE communication is ongoing, i.e., if no central device is connected, there are still BLE operations scheduled, since the BLE peripheral is required to transmit advertisement packets. We therefore evaluate the PRR in non-connected mode at different advertisement intervals. The results are similar to the earlier measurements and shown in Fig. 6.12.

During both experiments, the PRR of the BLE communication is constantly at 100% and is thus omitted in the evaluation plots.

6.3 Demonstrator

In order to show the functionality of the smart home CTC demonstrator described in Sec. 5.1 and Sec. 5.5, we recorded a video showing, that it is possible to conveniently control smart home devices using CTC. The latter can be seen in Fig. 6.13.

The video is available on `https://github.com/hanuka24/ctc-smarthome`.

Figure 6.13: Devices used in the CTC smart home demonstrator: a Nexus 6P smartphone, an IKEA Trådfri light bulb and a Danalock V3 door lock.

Chapter 7

Conclusions & Future Work

In this chapter, we give a summary of the thesis contributions and outline the evaluation results in Sec. 7.1. Finally, in Sec. 7.2 and overview of open challenges and future work is given.

7.1 Conclusion

In this thesis, we extend the cross-technology communication mechanism X-Burst to support Wi-Fi devices (i.e., a Raspberry Pi 3B+ and a Nexus 6P smartphone). We can thereby show that a communication between the three most popular wireless technologies in the 2.4 GHz ISM band is possible. In particular, X-Burst allows BLE, ZigBee and Wi-Fi devices to exchange data in a seamless and bidirectional manner, despite incompatible physical layers and without the need of a costly multi-radio gateway. Furthermore, X-Burst is applied on two off-the-shelf smart home devices, namely: a ZigBee-based IKEA Trådfri light bulb and a BLE-based Danalock V3 door lock. The CTC mechanism is implemented such that a seamless coexistence between an existing communication stack and CTC is possible, i.e., such that devices can operate using ZigBee/BLE and CTC simultaneously. This capability opens up the opportunity to provide legacy devices with CTC functionality without breaking existing features.

Our CTC implementation, based on packet-level modulation, can achieve data rates of more than 1 kbit/s, which is sufficient for the typical requirements of a smart home, as only small amounts of data have to be transmitted commonly (e.g., to control lamps or to get sensor data). Nonetheless, compared to native communication technologies, the throughput is limited and as the time-on-air for each packet is rather long, the influence of external interference may be substantial. The transmission speed and reception capabilities are highly dependent on the employed hardware and can thus be maximized using appropriate platforms. During the evaluation process, a promising communication range of at least 15 m for all considered devices was observed, which is sufficient for in-home applications. Furthermore, the memory footprint of the CTC implementation is minimal and allows the deployment on resource-constrained devices.

While existing work on CTC focuses on novel techniques maximizing the throughput and is mostly confined to academia, this thesis presents a concrete smart home application using real-world platforms and is a first step towards the actual deployment of CTC in a

broader scope. Although kept rather simple and small, the demonstration setup gives a glance of the potential of CTC in networks consisting of several different heterogeneous devices, which in particular applies to smart homes. We could show, that thanks to its general and modular design, X-Burst is feasible for a variety of different devices. Furthermore, its implementation is non-invasive, i.e., no hardware modifications are required. A simple firmware update can bring CTC capabilities to legacy devices and even allows to keep the existing communication functionality in place.

Yet, still a prototype, there are several open points which can be addressed in future work and are discussed in the following.

7.2 Future Work

Improving X-Burst support on Wi-Fi devices. In the current implementation, as shown in the evaluation, Wi-Fi devices and the Nexus 6P in particular, have the worst CTC performance with regard to throughput and communication range. Improvements in RSS sampling rates, alternative reception approaches (e.g., using Wi-Fi's Channel State Information (CSI)), as well as decreasing the *radio preparation time* would be beneficial in order to achieve a higher throughput and mitigate the influence of external radio interference, as the time-on-air can be reduced. It would further be of interest to divide a Wi-Fi channel into several subcarriers. This way, not only the sensitivity to external disturbances is decreased, but the generated interference on other ongoing traffic is reduced.

Improving X-Burst in general. In order to increase the robustness of the CTC mechanism, the implementation of encoding schemes tailored to X-Burst is of interest. The application of error-correcting codes could help to recover from transmission errors. Another possibility would be to adapt the encoding scheme to the data, i.e., to map frequently used symbols to short durations and use long durations for rarely used symbols only. In general, however, X-Burst's performance mainly depends on (and is often limited by) the hardware characteristics of the participating devices. Currently, the operation properties (e.g., alphabet and preamble) are derived manually and have to be shared among the devices at compile time. The computation of the 'fastest' possible configuration during run-time would help to automatically maximize the throughput for each set of devices.

Beyond X-Burst: Design of a smart home application layer. Cross-technology communication helps to overcome incompatibilities regarding wireless technologies. To allow a gateway-free, simple and convenient smart home setup as envisioned in the introduction, an application layer is required to allow seamless interaction between various smart home devices. Furthermore, a device discovery mechanism and addressing scheme is necessary to automatically add new devices to the network. CTC holds also potential for other applications besides smart homes. The possibility of direct data exchange between incompatible devices allows the implementation of channel-coordination schemes to mitigate cross-technology interference, which is especially of interest in the crowded 2.4 ISM GHz frequency band.

Security is a necessity. An important step towards the actual deployment of CTC-enabled devices and networks, is the implementation of security mechanisms. As any device on the same channel can participate in CTC communication, encryption is important to avoid eavesdropping, or even worse, the external, unwanted control of the CTC-enabled smart home appliance. A possible scheme could be a pairing process similar to the *Touchlink* commissioning used in ZigBee, where the devices have to be held very close to exchange keys and to establish a secure connection. In BLE and ZigBee stacks, encryption primitives are implemented, as both protocols provide the possibility for secure communication. Hence, their encryption algorithms could possibly be reused to secure CTC.

Bibliography

[1] Ghidra. A software reverse engineering (SRE) suite of tools developed by NSA's Research Directorate in support of the Cybersecurity mission https://ghidra-sre.org/.

[2] HUAWEI Nexus 6P. https://consumer.huawei.com/uk/support/phones/nexus-6p/.

[3] IKEA Trådfri GU10. https://www.ikea.com/gb/en/p/tradfri-led-bulb-gu10-400-lumen-wireless-dimmable-warm-white-60420041/.

[4] Nexmon https://github.com/seemoo-lab/nexmon.

[5] nRF5 SDK. https://www.nordicsemi.com/Software-and-tools/Software/nRF5-SDK. Accessed: 2020-08-05.

[6] QPython - Python on Android https://www.qpython.com/.

[7] Radiotap https://www.radiotap.org/.

[8] Raspberry Pi 3 Model B+. https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[9] Scapy. Packet crafting for Python2 and Python3 https://scapy.net/.

[10] Wi-Fi: Overview of the 802.11 Physical Layer and Transmitter Measurements.

[11] Smart Home Compatibility: Towards an open Eco System? https://www.hestiamagazine.eu/smart-home-compatibility-towards-an-open-eco-system, 2019. Accessed: 2020-07-07.

[12] The Greatest Barrier to 'Smart Home' Adoption is Complexity, New Study Shows. https://www.strata-gee.com/the-greatest-barrier-to-smart-home-adoption-is-complexity-new-study-shows/, 2019. Accessed: 2020-07-07.

[13] EmberZNet PRO Zigbee® Protocol Stack Software. https://www.silabs.com/products/development-tools/software/emberznet-pro-zigbee-protocol-stack-software, 2020. Accessed: 2020-07-27.

[14] ZigBee Alliance. https://zigbeealliance.org/, 2020. Accessed: 2020-07-27.

[15] M. Brown. Apple, Google and Amazon's plan could finally end the smart home nightmare. `https://www.inverse.com/article/61830-apple-google-and-amazon-s-smart-home-standard`.

[16] H. Brunner. Enabling Bidirectional Cross-Technology Communication on Off-The-Shelf Wi-Fi Devices, 2019. Seminar Project.

[17] H. Brunner, R. Hofmann, M. Schuß, J. Link, M. Hollick, C. A. Boano, and K. Römer. Demo: Cross-Technology Broadcast Communication between Off-The-Shelf Wi-Fi, BLE, and IEEE 802.15.4 Devices. In *Proceedings of the 2020 International Conference on Embedded Wireless Systems and Networks*, EWSN '20, page 176–177, USA, 2020. Junction Publishing.

[18] K. Chebrolu and A. Dhekne. Esense: Energy Sensing-Based Cross-Technology Communication. *IEEE Transactions on Mobile Computing*, 12(11):2303–2316, Nov. 2013.

[19] Z. Chi, Y. Li, H. Sun, Y. Yao, Z. Lu, and T. Zhu. B2W2: N-Way Concurrent Communication for IoT Devices. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, SenSys '16, page 245–258, New York, NY, USA, 2016. Association for Computing Machinery.

[20] Daintree Networks. Getting Started with ZigBee and IEEE 802.15.4. `https://www.coursehero.com/file/20658485/Zigbee-GettingStarted/`, 2008. Accessed: 2020-07-29.

[21] Danalock International ApS. Danalock V3 Smart Lock. `https://danalock.com/products/danalock-v3-smart-lock/`.

[22] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN '04, page 455–462, USA, 2004. IEEE Computer Society.

[23] K. D. Foote. A Brief History of the Internet of Things. `https://www.dataversity.net/brief-history-internet-things/`, 2016. Accessed: 2020-07-07.

[24] C. Gavrila, V. Popescu, M. Fadda, M. Anedda, and M. Murroni. On the Suitability of HbbTV for Unified Smart Home Experience. *IEEE Transactions on Broadcasting*, pages 1–10, 2020.

[25] F. Gringoli, M. Schulz, J. Link, and M. Hollick. Free Your CSI: A Channel State Information Extraction Platform For Modern Wi-Fi Chipsets. In *Proceedings of the 13th International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, WiNTECH '19, page 21–28, New York, NY, USA, 2019. Association for Computing Machinery.

[26] X. Guo, X. Zheng, and Y. He. WiZig: Cross-technology energy communication over a noisy channel. In *Proceedings of IEEE INFOCOM*, pages 1–9, 2017.

[27] J. Hendrickson. How to Control Your Entire Smarthome Through One App. `https://www.howtogeek.com/435765/how-to-control-your-entire-smarthome-through-one-app/`, 2019. Accessed: 2020-07-07.

[28] R. Hofmann. X-Burst: Cross-Technology Communication for Off-the-Shelf IoT Devices. Master's thesis, Graz University of Technology, 2018.

[29] R. Hofmann, C. A. Boano, and K. Römer. X-Burst: Enabling Multi-Platform Cross-Technology Communication between Constrained IoT Devices. In *Proceedings of the* 16^{th} *IEEE International Conference on Sensing, Communication and Networking (SECON)*, pages 1–9. IEEE, 2019.

[30] W. Jiang, Z. Yin, R. Liu, Z. Li, S. M. Kim, and T. He. BlueBee: A 10,000x Faster Cross-Technology Communication via PHY Emulation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, New York, NY, USA, 2017. Association for Computing Machinery.

[31] J. Kastrenakes. It's really complicated to connect the Home of the Future. `https://www.theverge.com/2018/8/20/17724278/smart-home-installation-setup-complicated-home-of-the-future-grant-imahara`, 2018. Accessed: 2020-07-07.

[32] S. M. Kim and T. He. FreeBee: Cross-Technology Communication via Free Side-Channel. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, page 317–330, New York, NY, USA, 2015. Association for Computing Machinery.

[33] A. Koubaa, M. Alves, and E. Tovar. IEEE 802.15.4 for Wireless Sensor Networks: A Technical Overview. Polytechnic Institute of Porto, Technical Report TR-050702, 2005.

[34] Z. Li and T. He. WEBee: Physical-Layer Cross-Technology Communication via Emulation. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, page 2–14, New York, NY, USA, 2017. Association for Computing Machinery.

[35] K. L. Lueth. IoT 2019 in Review: The 10 Most Relevant IoT Developments of the Year. `https://iot-analytics.com/iot-2019-in-review/`, 2020. Accessed: 2020-07-07.

[36] Meticulous Research. Top 10 Companies in Smart Home Market. `https://meticulousblog.org/top-10-companies-in-smart-home-market/`, 2020. Accessed: 2020-08-22.

[37] M. Moazzami, G. Xing, D. Mashima, W. Chen, and U. Herberg. SPOT: A smartphone-based platform to tackle heterogeneity in smart-home IoT systems. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 514–519, 2016.

[38] Nordic Semiconductors. Poly-Control Aps employs nRF52832 to create retrofit smart door lock. `https://www.nordicsemi.com/news/2017/10/poly-control%20aps%20danalock%20v3%20smart%20lock`, Accessed: 2020-08-03.

[39] NXP Semiconductors. *ZigBee Light Link User Guide*, 2016.

[40] NXP Semiconductors. *ZigBee Cluster Library (for ZigBee 3.0) User Guide*, 2018.

[41] M. Schulz, F. Gringoli, D. Steinmetzer, M. Koch, and M. Hollick. Massive Reactive Smartphone-Based Jamming Using Arbitrary Waveforms and Adaptive Power Control. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '17, page 111–121, New York, NY, USA, 2017. Association for Computing Machinery.

[42] M. Schulz, D. Wegemer, and M. Hollick. Nexmon: Build Your Own Wi-Fi Testbeds With Low-Level MAC and PHY-Access Using Firmware Patches on Off-the-Shelf Mobile Devices. In *Proceedings of the 11th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, WiNTECH '17, pages 59–66, New York, NY, USA, 2017. ACM.

[43] M. Schuß, C. A. Boano, M. Weber, M. Schulz, M. Hollick, and K. Römer. JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controllable and Repeatable Wi-Fi Interference. In *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks*, EWSN '19, page 83–94, USA, 2019. Junction Publishing.

[44] Silicon Labs. *UG250: Thunderboard Sense User's Guide*, 2017.

[45] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund. Industrial Internet of Things: Challenges, Opportunities, and Directions. *IEEE Transactions on Industrial Informatics*, PP:1–1, 2018.

[46] M. Spörk, C. A. Boano, M. Zimmerling, and K. Römer. BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, New York, NY, USA, 2017. Association for Computing Machinery.

[47] B. Stottelaar. Hacking the IKEA TRÅDFRI. `https://github.com/basilfx/TRADFRI-Hacking`, 2020. Accessed: 2020-07-27.

[48] Texas Instruments. SimpleLink CC2650 wireless MCU LaunchPad Development Kit. `https://www.ti.com/tool/LAUNCHXL-CC2650`.

[49] K. Townsend. *Getting Started with Bluetooth Low Energy*. O'Reilly, 2014.

[50] WIRED Brand Lab. Why the Typical Smart Home Has Some Growing Up To Do. `https://www.wired.com/brandlab/2018/10/typical-smart-home-growing/`. Accessed: 2020-07-07.

[51] Z. Yin, Z. Li, S. M. Kim, and T. He. Explicit Channel Coordination via Cross-Technology Communication. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '18, page 178–190, New York, NY, USA, 2018. Association for Computing Machinery.

[52] Z. Yu, C. Jiang, Y. He, X. Zheng, and X. Guo. Crocs: Cross-Technology Clock Synchronization for WiFi and ZigBee. In *Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*, EWSN '18, page 135–144, USA, 2018. Junction Publishing.

[53] X. Zhang and K. G. Shin. Gap Sense: Lightweight coordination of heterogeneous wireless devices. In *Proceedings of IEEE INFOCOM*, pages 3094–3101, 2013.

[54] Zolertia. *Zolertia Firefly Revision A2 Internet of Things hardware development platform, for 2.4-GHz and 863-950MHz, IEEE 802.15.4, 6LoWPAN and ZigBee Applications*, 2017.