David Grubmair, BSc.

X-Sync: Accurate Cross-Technology Clock Synchronization between BLE and IEEE 802.15.4 Devices

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

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

Advisor: Dipl.-Ing. Rainer Hofmann

Graz, August 2020

David Grubmair, BSc.

X-Sync: Accurate Cross-Technology Clock Synchronization between BLE and IEEE 802.15.4 Devices

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Telematik

eingereicht an der

Technischen Universität Graz

Betreuer

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

Institut für Technische Informatik

Zweitbetreuer: Dipl.-Ing. Rainer Hofmann

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

Abstract

Clock synchronization is an essential service for wireless sensor networks, as it allows to correctly interpret and fuse consistent sensor data, as well as to achieve a high energy efficiency and localization accuracy. Whilst several techniques have been proposed in the literature to achieve an accurate clock synchronization in a low-power wireless sensor network, the focus has always been on *homogeneous* networks, i.e., on networks composed of devices making use of the same wireless technology and hence employing compatible physical layers (PHYs). With the advent of the Internet of Things (IoT), a plethora of different wireless technologies have been designed and commercialized in order to cover the diverse requirements of different IoT application domains. This results in *heterogeneous* networks making use of different technologies (e.g., BLE, IEEE 802.15.4, Wi-Fi), as well as in an increased usage of multi-radio gateways. The latter are needed to coordinate the operations of heterogeneous devices, to distribute information, as well as to collect and timestamp measurements. Such multi-radio gateways, however, are expensive, inflexible, and represent a single point of failure.

In this Master thesis we present X-Sync, a novel cross-technology clock synchronization (CTCS) scheme that allows to seamlessly exchange timestamps between IEEE 802.15.4 and BLE devices as well as to synchronize their clocks with microsecond level accuracy. X-Sync enables such an accurate clock synchronization despite the use of packet-level cross-technology communication (CTC), which is characterized by unpredictable delays, low data rates and error prone transmissions. To overcome these challenges, X-Sync adapts and extends a packet-level CTC scheme, named X-Burst, to allow timestamping at the medium access control layer, and provides novel features to compensate the delays added by the use of CTC. X-Sync was implemented on the Contiki-NG operating system for different off-the-shelf hardware platforms to show its generality and portability. These platforms include the TI CC2650 LaunchPad and Sensortag, the TelosB node, as well as the Zolertia Firefly. An experimental evaluation shows that X-Sync can achieve a bidirectional synchronization between these off-the-shelf IEEE 802.15.4 and BLE devices with an accuracy as low as $\pm 125\mu s$ (CC2650, Firefly, and TelosB configured as transmitter) and $\pm 4ms$ (TelosB configured as receiver) when using an update rate of 60 seconds.

Kurzfassung

Taktsynchronisation bezeichnet einen essentiellen Dienst in drahtlosen Netzwerken, der erlaubt Sensordaten zusammenzufügen und zu interpretieren. Es wurden in der Literatur verschiedene Techniken vorgestellt, die eine präzise Synchronisierung erlauben, doch der Fokus lag immer auf *homogenen* Netzwerken. Also ein Netz aus Geräten, die dieselbe Funktechnologie verwenden. Mit dem Aufkommen des „Internet of Things" (IoT) wurden viele verschiedene Funktechnologien entwickelt und kommerzialisiert, um diversen Anforderungen von IoT – Anwendungsgebieten gerecht zu werden. Die Folge waren einerseits *heterogene* Netzwerke, die unterschiedliche Funktechnologien kombinieren (zum Beispiel BLE, IEEE 802.15.4, Wi-Fi) und andererseits ein erhöhter Bedarf an Gateways, die mit mehreren integrierten Funktechnologien ausgestattet sind. Diese werden verwendet um alle Teilnehmer des heterogenen Netzwerks zu koordinieren, Informationen zu verteilen, Messergebnisse zu sammeln und mit einem Zeitstempel zu versehen. Diese Gateways mit mehreren integrierten Funktechnologien sind jedoch teuer, unflexibel und stellen einen einzelnen Ausfallspunkt dar.

In dieser Arbeit präsentieren wir X-Sync, ein neuartiges, technologieübergreifendes Taktsynchronisierungsschema, welches das Austauschen von Zeitstempel zwischen IEEE 802.15.4 und BLE Geräten erlaubt. Dadurch können deren interne Uhren mit einer Genauigkeit im Mikrosekundenbereich synchronisiert werden. Dafür wird eine paket-basierte, technologieübergreifende Kommunikationsmethode verwendet, die durch unvorhersehbare Verzögerungen, langsame Übertragungsraten und störanfällige Übertragungen charakterisiert wird. Deswegen basiert X-Sync auf X-Burst, einer technologieübergreifenden Kommunikationsmethode. Dies ermöglicht eine Synchronisierung auf der MAC (medium access control) Schicht und erlaubt das Kompensieren von Verzögerungen. X-Sync wurde auf Basis des Contiki-NG Betriebssystems implementiert. Um die Portabilität zu zeigen, wurden verschiedene Entwicklungsplattformen verwendet, wie TI CC2650 LaunchPad and Sensortag, TelosB und Zolertia Firefly. Die Evaluierung zeigt das X-Sync wechselseitige Synchronisation zwischen IEEE 802.15.4 und BLE Plattformen mit einer Genauigkeit von $\pm 125\mu s$ (CC2650, Firefly und TelosB konfiguriert als Sender) und $\pm 4ms$ (TelosB konfiguriert als Empfänger) bei einem Synchronisationsintervall von 60s ermöglicht.

Contents

List of Figures

List of Tables

1 Introduction

The Internet of Things (IoT) connects everyday objects (things) such as sensors, vehicles and household appliances to other objects or their respective backends over the Internet and allows monitoring, detection and controlling of these devices [1]. The areas of application are thereby widespread and include smart cities, industry 4.0, connected cars, as well as consumer IoT applications such as wearables and smart home devices [2]. With an increasing number of deployed IoT systems in these and other application domains, the number of interconnected IoT devices is expected to grow continously and reach 75 billions by 2025 [3].

A fundamental building block to create IoT systems that sense the environment and connect to each other and to the Internet are Wireless Sensor Networks (WSN). These networks are typically self-organizing and connected in a multi-hop fashion [4]. WSNs differ from conventional networks, as they focus on low-cost, battery-powered nodes and as they are characterised by unreliable links and computationally constrained nodes. Apart from that, the number of nodes is very high compared to a classical (wireless) network.

Clock synchronization is hereby an essential aspect for WSNs [5]. It plays a very important role in the interpretation of sensor data, fusion of (consistent) measurements, energy efficiency, and localisation. Apart from that, it is crucial for the correct and efficient operation of network protocols [6] and management tools [7]. For example, stale and inconsistent measurements within a power grid may result in erroneous control that endangers the grid's safety [8]. De-synchronization of consumer smart meters may as well lead to instability of the real-time energy market [9].

Another example for the need of a tight clock synchronization in an industrial environment is a large-scale manufacturing line. Sensors have to be deployed on each segment, continuously delivering status updates. These sensors have to be precisely synchronized to each other for the control unit to enable specific production steps at the right time. Synchronization failures can, for example, cause clashes of robot arms and disrupt the production pipeline.

1.1 Problem Statement

Increasing heterogeneity of wireless devices. With the advent of embedded and cyber-physical systems, different wireless technologies with incompatible physical layers were developed to satisfy largely diverse application requirements [10]. While some technologies focus on very high data rates e.g., Wi-Fi, others focus on energy consumption, e.g., IEEE 802.15.4 and Bluetooth Low Energy (BLE).
Wireless sensor systems are typically based on homogeneous networks (e.g., composed of devices making all use of a BLE radio), where every node implements the same physical radio layer, and can hence send and receive messages on the shared radio channel. Examples of such homogeneous networks are volcanic eruption monitoring [11], and health monitoring [12].

Monitoring volcanic eruptions is hereby an application for homogeneous wireless sensor networks. The authors use infrasonic sensor modules to sample low-frequency acoustic signals which are then transmitted using ZigBee to a multi-radio gateway. This gateway translates the received packets and transmits them over a long distance to a base station. Based on this data, the authors created an algorithm which detects for local explosions. This only works if the sensor readings are synchronized to each other, which is achieved by using a clock synchronization protocol with one central time server equipped with a GPS reference clock. Another example of a homogeneous application is described in the paper "Health Monitoring of civil infrastructures using WSN" [12]. In this application, the sensor boards sample the instantaneous acceleration with a high precision sensor utilizing a high sampling frequency. The clocks on these sensors are synchronized to each other using a standard clock synchronization protocol. The sensor boards use a (homogeneous) ZigBee network to send the measurement data to a base station, where it is stored and analysed.

Recent papers show that the requirements of WSNs are becoming more diverse [13], [14]. Depending on the application, several different WSNs using various technologies have been deployed in the same area, which results in "islands" of nodes employing different PHY layers operating autonomously and being orchestrated by means of multi-radio gateways

[15], [16]. The use of multi-radio gateways actually represents the state-of-the-art solution to synchronize heterogeneous networks.

The authors of [13] develop such a heterogeneous WSN for smart home and building automation. They use three different wireless technologies (Wi-Fi, ZigBee and BLE) to create a multi-hop network. This network was evaluated by implementing a building automation system prototype which controls and manages air-conditioners and AC power meter units. Another recent paper evaluates a multi-radio ZigBee/BLE gateway for remote parameter monitoring [14]. A Linux gateway is used to implement a low-cost platform which communicates with both the BLE and the ZigBee networks and provides access to the Internet. SSL and DTLS connections are then used to establish a connection to the cloud.

This heterogeneous networking approach can be seen in Figure 1.1, which conceptually shows two heterogeneous networks where devices directly exchange information (e.g., BLE and IEEE 802.15.4). For simplicity, only one-hop communication with a small number of nodes is sketched. In reality, a wireless sensor network can inherit a multi-hop system with complex routing requirements. The used multi-radio gateway embeds both radios (i.e., a BLE and an IEEE 802.15.4 radio) to dynamically translate the received packets from one technology to the other. Apart from the additional hardware costs, this effectively introduces a single point of failure, increases network traffic, and leads to higher end-to-end delays.

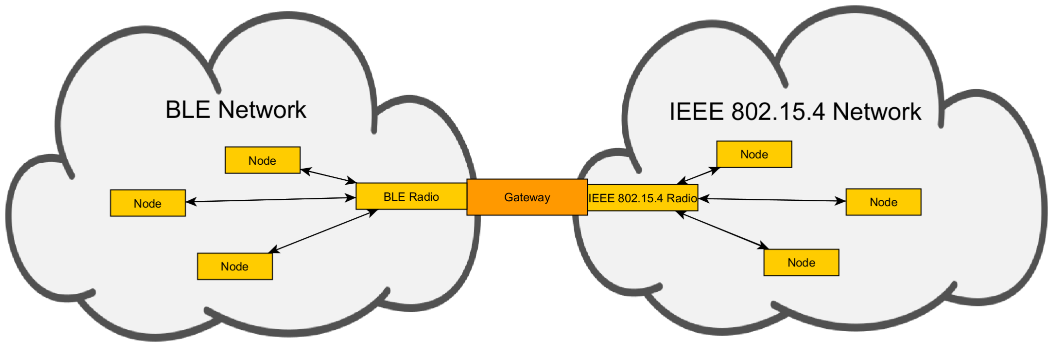


Figure 1.1: Two heterogeneous networks (e.g., BLE and IEEE 802.15.4) where devices exchange information through a multi-radio gateway.

Communication between devices with incompatible physical layers is also possible by using *cross-technology communication* (CTC) schemes, which provide an alternative to the use of gateways. Cross-technology communication is a new paradigm that consists in transmitting data directly from one physical layer to another without the need of additional hardware.

One of the first papers in the field of CTC was E-Sense, a communication framework based on energy sensing [17]. It enables unidirectional communication from Wi-Fi to ZigBee devices by encoding information into the duration of energy bursts. Hereby, each energy burst signature represents a specified information such as a predefined message, command, or value and thus does not allow an arbitrary information exchange. Another early work in the field of CTC was GapSense, a lightweight coordination mechanism for heterogeneous wireless devices [18]. Hereby, a preamble is defined and, upon detection of this preamble, receivers can react accordingly. Applications of that mechanism include collision avoidance (in both time and frequency domain) and clock rate management. Like E-Sense, GapSense also does not allow arbitrary data transmissions, but only predefined symbols. Another noteworthy paper in this field is about cooperative carrier signalling, which aims to harmonize the coexistence of Wi-Fi and ZigBee devices [19]. This novel mechanism significantly reduces collisions of both networks. It works by sending a busy tone from a ZigBee node to the nodes in the Wi-Fi network. These nodes can sense this information and adjust their behaviour accordingly. No arbitrary data can be sent and the communication is only unidirectional.

State-of-the-art CTC approaches can be divided into two categories: PHY layer emulation schemes and packet-level schemes [20]. The former work by using the modulation scheme similarities, defined in the PHY layer, between radios operating on the same frequency band. By using analytical calculations, a mapping alphabet can be found which allows the emulation of packets of another PHY layer. For example, BlueBee is an influential work in this field allowing the emulation of ZigBee [1] packets within a valid BLE frame [21]. Using BlueBee, the IEEE 802.15.4 device is then able to decode the transmitted packet and interpret the data. However, if the two

[1]The terms Zigbee and IEEE 802.15.4 will be used interchangeably throughout this thesis.

used wireless physical layers are fundamentally different, finding a suitable mapping alphabet gets more difficult, if not impossible.

Packet-level CTC schemes work by sending and sensing energy durations or energy levels. This is achieved by sending valid packets onto the shared physical channel with varying lengths or transmission power. The receiver has to be able to sense the signal strength on the channel and recover the encoded duration or energy level.

With the use of CTC, the example given in Figure 1.1 could be replaced by the node arrangement shown in Figure 1.2. Any node could now directly communicate with any other node in the heterogeneous network without the need for a gateway. In this example, we assume that each network has one dedicated node carrying out CTC communication.

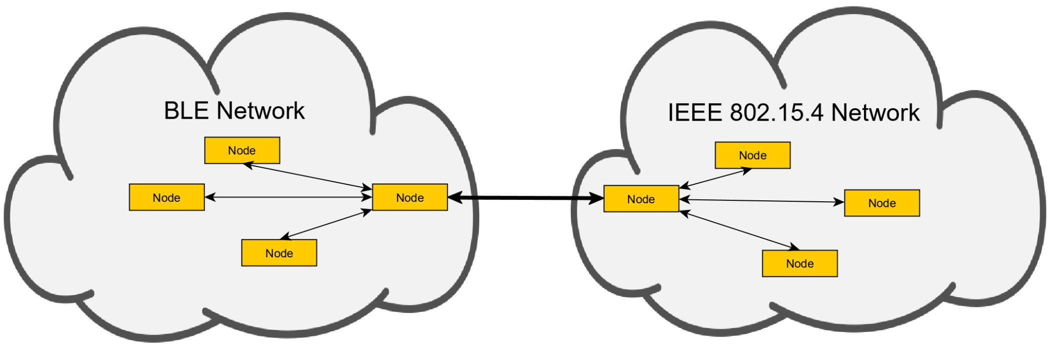


Figure 1.2: Two heterogeneous networks (e.g., BLE and IEEE 802.15.4) where devices directly exchange information without the need of a multi-radio gateway.

Building upon such a network configuration, we assume that we have a master device which acts as a clock reference for all nodes in both heterogeneous networks. In order to achieve that, the synchronization process can be divided in two steps, as shown in Figure 1.3. First, the two nodes carrying out CTC (called "border nodes") are synchronized using cross-technology clock synchronization. Second, within each network, the nodes autonomously synchronize to the clock of the border node using standard clock synchronization schemes. If, for whatever reason, the border node fails, the network could elect a new border node, because each node in the network is capable of carrying out CTCS. This eliminates the single point of failure of conventional solutions based on a multi-radio gateway. In addition to that, the end-to-end delay can be significantly lowered and no additional hardware costs are necessary.

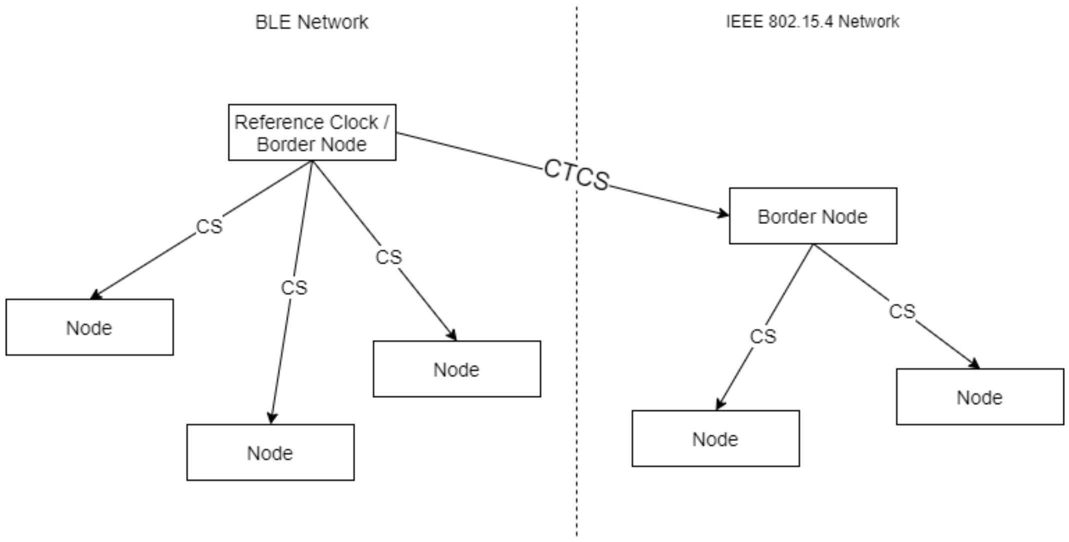


Figure 1.3: Synchronization of all nodes in a heterogeneous network, assuming that a master device acts as a reference clock for all nodes. First, the two border nodes are synchronized using CTCS (in this case to the BLE master). Second, within each network, the nodes autonomously synchronize to the clock of the border node using standard clock synchronization (CS) schemes. Note that the border node of the IEEE 802.15.4 network can also act as reference clock.

1.2 Contributions

To allow high accuracy cross-technology clock synchronization between wireless devices with incompatible physical layers, we extend and revise an existing cross-technology communication scheme, called **X-Burst** [20]. X-Burst is a portable framework that allows multiple constrained IoT platforms with diverse characteristics to seamlessly interact using packet-level CTC. In their original paper [20], the authors explicitly target the BLE and the IEEE 802.15.4 physical radio layers due to their widespread use in IoT systems. However, due to its generality, X-Burst is not limited to these communication schemes and can also be used with other wireless standards [22]: this is possible as the scheme only makes use of energy sensing features available in standard compliant off-the-shelf devices.

Due to the nature of packet-based CTC, indeterministic delays are introduced, which greatly reduces the achievable synchronization accuracy. We explore methods to compensate these delays and consequentially allow the transmission and reception of cross-technology timestamps with microsecond accuracy. Additionally, we explore ways to improve the synchronization accuracy by applying filtering and clock skew estimation algorithms. Furthermore, we have integrated our solution into the Contiki-NG operating system and evaluated it on various off-the-shelf development platforms. In summary, our contributions are 5-fold:

Design and development of a cross-technology clock synchronization scheme. In this thesis we have developed an extension for the X-Burst cross-technology communication scheme which allows transmission of timestamps in the medium access control (MAC) layer of BLE and IEEE 802.15.4 devices. MAC layer timestamping is hereby needed to remove the indeterministic delays present in radio transmissions. We call this cross-technology clock synchronization scheme **X-Sync**, which allows accurate clock synchronization in heterogeneous wireless networks without the need of a multi-radio gateway.

Exploring ways to compensate the non-deterministic delays present in packet-level CTC transmissions. Due to the packet-level nature of X-Burst, non-deterministic delays are introduced, which are highly varying and too big to neglect to achieve microsecond-level accuracy. To compensate these delays, X-Sync defines software algorithms to detect and compensate the various delays, which is essential to achieve an accurate synchronization.

Exploring ways to increase the relative synchronization accuracy between a transmitting node and one (or many) receiving nodes. The notion of time at two independent clocks will always be different. This is called clock skew and can be linearly approximated, assuming short-term and constant environmental conditions. Accounting for that reduces the number of synchronization packets needed for a given accuracy and therefore reduces the overall energy consumption of the connected system. We explore and evaluate the use of two clock skew estimation algorithms, linear regression and random sample consensus (RANSAC), in a cross-technology context. As packet-level CTC schemes inherently have a higher susceptibility to disturbances on the used radio channel, the use of filtering algorithms such as RANSAC allows the detection of outliers and a more accurate synchronization even in congested environments.

Integration of cross-technology clock synchronization into Contiki-NG. We integrate X-Sync into the Contiki-NG operating system while still maintaining interoperability with X-Burst. Contiki-NG introduces a generic layer on top of the used IoT hardware platform, which allows to write platform independent code for higher portability. However, due to the precise timing requirements, we also change the hardware specific code and optimize it for low-latency MAC timestamp transmission.

Evaluation on different hardware platforms. We evaluate X-Sync in terms of synchronization accuracy, preamble length, buffer size, synchronization accuracy, energy consumption, and long-term synchronization accuracy. Additionally, we evaluate the performance of different clock skew compensation algorithms for their application in a cross-technology context. We use the following off-the-shelf IoT development platforms to show the generality of this solution: the TI CC2650 LaunchPad and Sensortag [23], the Tmote Sky [24], and the Zolertia Firefly [25].

1.3 Thesis Structure

This thesis is structured as follows. In Chapter 2, background information regarding this thesis' context, clock synchronization, and cross-technology communications are given. Additionally, related works are compared, and the foundation of this work, X-Burst, is presented in detail. The last part of this section describes the employed hardware and software employed in this thesis.

In Chapter 3, we first define requirements for the sought solution, followed by an overview of X-Sync and its working principle. From these requirements, we derive the corresponding challenges and propose solutions for each of them.

The concepts of the previous chapter are then implemented in Chapter 4, which shows how X-Sync is divided into blocks for extensibility and to reduce the complexity of the solution. These blocks are then described in detail.

Chapter 5 describes the experimental setup and presents an evaluation about the accuracy, energy consumption, memory footprint and reliability of X-Sync. These findings are then combined and used for a long-term, real-world evaluation at the end of this chapter.

Finally, Chapter 6 summarizes the contributions of this thesis and gives a short outlook into possible future work.

2 Background

A short introduction to wireless sensor networks is given in Section 2.1, which also introduces the wireless technologies employed in this thesis, followed by an introduction to classical clock synchronization systems in homogeneous networks (Section 2.2). Section 2.3 distinguishes between packet-level and physical layer emulation schemes for cross-technology communication and describes them. Thereafter, the working principle of X-Burst, which is the base for X-Sync, is explained in Section 2.4. Albeit CTCS being a relatively new research area, the related works in this field are summarized in Section 2.5. Finally, the employed off-the-shelf hardware platforms (Section 2.6), as well as the employed software (Section 2.7) are discussed.

2.1 Wireless Sensor Networks

The Internet of Things represents the idea of expanding the Internet to the real world, embracing everyday objects. Physical items are now connected to the Internet as access points, sharing and acting upon information [26]. As the number of connected devices is continuously increasing, maintaining and creating a wired infrastructure is an obstacle and makes the use of wireless technologies compulsory for the number of nodes in a typical IoT network [27]. Wireless sensor networks solve this problem by providing a way to connect thousands of nodes to the Internet or to each other. Application designers can choose from a variety of wireless technologies with varying specifications, costs, and functional properties.

Specialized wireless technologies have been developed or adapted for the use in IoT networks, with the most prominent ones being Bluetooth Low Energy (BLE) and IEEE 802.15.4 (ZigBee). Other relevant wireless technologies in this context include, but are not limited to, LoRa/LoraWAN [28] and many vendor specific protocols like LowPowerLabs [27]. The properties of a wireless technology are greatly influenced by the used carrier frequency and the specified modulation scheme. Apart from the fact that only some frequency bands are licensed for the use in IoT networks, lower carrier frequencies result in lower data rates and higher maximum ranges. In this thesis, we specifically focus on BLE and IEEE 802.15.4 radios operating in the 2.4 GHz frequency band, which are the most widespread technologies in today's IoT products.

2.1.1 IEEE 802.15.4

ZigBee is one prominent example for low-rate WPAN that builds upon the IEEE 802.15.4 standard [29], which describes the MAC and the physical layer of the network stack [30]. Other standards building upon IEEE 802.15.4 include ISA100.11a [31], WirelessHART [32], 6LoWPAN [33] and many more. For the purpose of this thesis, only these two lower layers are relevant. IEEE 802.15.4 is designed for low-power, energy-efficient and battery operated usage in networks with over thousands of nodes in one network. This is achieved by allowing nodes to turn off their radios during inactivity. The power consumption during radio activity is relatively high compared to the consumption during CPU activity. The typical way to extend the battery life of nodes is to keep the active radio time as short as possible. This is normally achieved by synchronizing the nodes to each other by defining active and sleep times, often referred to as "duty-cycling" technique. The original physical layer standard (2003) defines three different usable unlicensed frequency bands:

- 868.0–868.6 MHz: Licensed in Europe, allows one channel.
- 902–928 MHz: Licensed in North America, allows thirty channels.
- 2400–2483.5 MHz: Licensed for worldwide use, up to sixteen available channels, numbered from 11 to 26.

Figure 2.1 shows the spectrum of IEEE 802.15.4 radios in the 2.4 GHz band. The 3 MHz gaps between the channels are needed because of the use of direct sequence spread spectrum as modulation scheme. In the crowded 2.4 GHz band, a multitude of wireless technologies operate due to its classification as industrial, scientific and medical (ISM) band.

The IEEE 802.15.4 standard further defines a carrier sense multiple access mechanism to lower the amount of collisions on the channel. Before a transmission will be started, the activity on the intended channel is measured

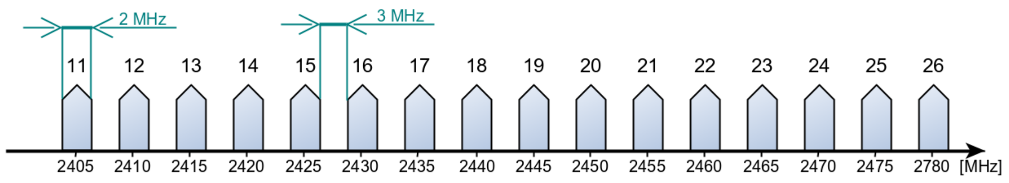


Figure 2.1: Frequency spectrum of IEEE 802.15.4 in the 2.4 GHz band.

by retrieving the radio signal strength information (RSSI). This value will measure the current activity on the chosen channel in dBm and is compared against a clear channel assessment (CCA) threshold. According to the IEEE 802.15.4 standard [29], the radio will apply a moving window averaging filter before returning the requested energy information. We will refer to this property as averaging radios, according to the definition in [20]. According to the result, the radio will either delay the transmission and try again after a defined or random back-off period, or send the requested radio packet. To save energy, CCA is also used to wake-up from sleep mode.

Further specifications include data rates between 25 kBit/s and 250kBit/s depending on the chosen frequency band and the chosen configuration. Typical achievable ranges are between 10 and 100 meters, depending on the configured transmission power.

2.1.2 Bluetooth Low Energy

Bluetooth low energy (BLE), also sometimes referred to as Bluetooth Smart, is a standard for low-rate WPAN that focuses on low-energy applications. Due to this focus, it is not compatible with classical Bluetooth and thus many hardware manufacturers offer chips with a built-in dual-stack layout supporting both standards. BLE is operating with forty 2-MHz wide channels in the 2.4 GHz band. It offers a significant higher peak bitrate than the IEEE 802.15.4 standard with 1 MBit/s while still maintaining the 100 meter transmission range under ideal conditions. This is possible by using a Gaussian frequency shift keying modulation scheme. The frequency spectrum of BLE can be seen in Figure 2.2. In addition to data channels, three advertising channels are defined: these allow devices to broadcast advertising packets. This mechanism enables devices to easily connect to existing networks.

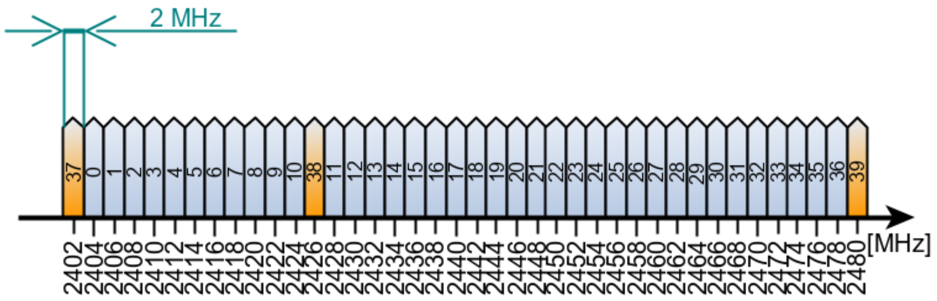


Figure 2.2: Frequency spectrum of BLE. The orange bands indicate the advertising channels.

Other than the IEEE 802.15.4 standard, BLE does not implement a form of clear channel assessment algorithm. Instead, it uses channel hopping and blacklisting to determine occupied channels. However, the most prominent BLE transceiver manufacturers (TI, Nordic Semiconductor and Cypress) include RSSI querying into their BLE radio ICs, which can be seen as a de-facto standard feature.

2.2 Clock Synchronization in Wireless Sensor Networks

Clock synchronization schemes in WSN can be divided into two layers building on top of each other, as seen in Figure 2.3. The timestamp transmission/reception layer provides a fundamental building block of any high level synchronization scheme as described in Section 2.2.1. The accuracy of the clock synchronization scheme is primarily defined in this layer. High-level clock synchronization schemes build on top of this layer, as shown in Section 2.2.2. These schemes can define methods to synchronize multi-hop networks, increase the synchronization accuracy, estimate the skew rate, and/or choose a reference node from all the nodes in the network.

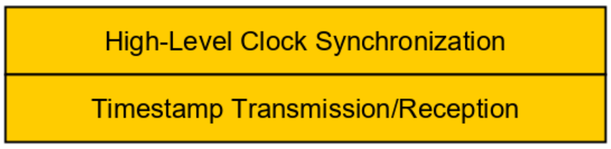

High-Level Clock Synchronization
Timestamp Transmission/Reception

Figure 2.3: Clock synchronization building blocks. Clock skew estimation algorithms are included in most high level clock synchronization schemes.

2.2.1 Timestamp Transmission/Reception

At the lowest architectural level, clock synchronization works by sending timestamps from a transmitting node to one (or many) receiving nodes. This fundamental primitive is used by high level synchronization protocols as a building block. The accuracy of the clock synchronization system is hence primarily limited by the delays introduced in this layer. Constant, deterministic delays can be compensated in higher layers, but varying, non-deterministic delays have a direct impact on the achievable accuracy. Figure 2.4 shows the critical path of a typical timestamp transmission.

Without the compensation of the delays shown in Figure 2.4, the accuracy of the clock synchronization is lowered significantly. Therefore, MAC timestamping can be used.

MAC Timestamping

MAC timestamping is based on appending timestamps on the lowest possible layer in the latest possible moment before transmission. This allows to avoid non-deterministic delays during synchronization. Upon reception, this timestamp will be processed as fast as possible and further decoded by the upper layers. In detail, MAC timestamping works as follows:

- Create and enqueue a new packet on node A with enough space reserved for the MAC timestamp.
- When the transmission is started, replace the reserved space with a generated timestamp T_1.
- As soon as the preamble is received on the receiving node B, generate a timestamp T_2

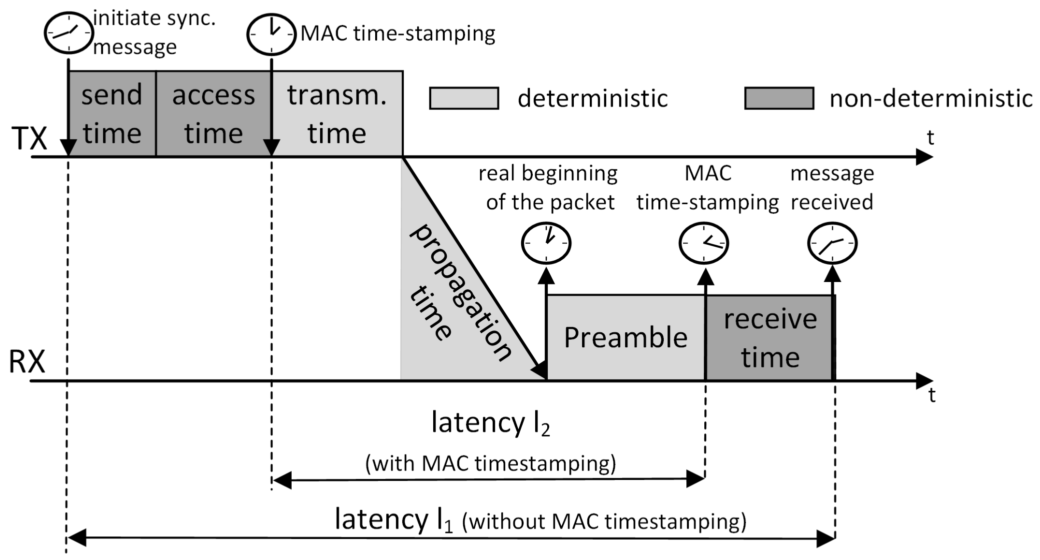


Figure 2.4: Critical path between sender and receiver with timespans divided into deterministic and non-deterministic delays. Latency l_1 shows the uncompensated delay between the initiation of the synchronization message at the transmitting node (TX) and the reception of the complete message at the receiving node (RX). With the use of MAC timestamping, non-deterministic delays can be avoided which is shown by latency l_2.

- After successful reception of the packet, node B has now a clock synchronization pair consisting of (T_1, T_2) that can be used for synchronization.

The transmission time, the propagation time, as well as the time to process the preamble cannot be compensated for within one transmission. High-level clock synchronization schemes can compensate for them by using round-trip synchronization, if they assume that both transmission paths have the same delays. This delays have hence certain properties:

- hardware specific;
- deterministic;
- predictable;
- direct proportional to the distance between transmitter and receiver;
- constant under the same environmental conditions;

2.2.2 High-Level Synchronization Schemes

High-level synchronization schemes make use of a timestamp transmission/reception building block with the goal of increasing the achievable accuracy and/or supporting multi-hop networks. In addition to that, these schemes often include algorithms to find the best suitable node acting as a time reference for the whole WSN. Skew rate estimation is an important mechanism to lower the overall energy consumption and increase the achievable accuracy and is explained in detail in this section. Followed by that, the most prominent high level synchronization schemes are listed and briefly described.

Clock Skew Compensation

All practical clocks are running at slightly different rates with respect to each other. The drift factor between two oscillators / clocks is called clock skew. Correcting this skew becomes especially important for long-term synchronization and has a major impact at the power consumption by keeping the number of synchronization packets low [5]. Assuming short-term constant environmental temperature conditions, this dependency is assumed to be linear [5]. Most high-level synchronization schemes already include compensation algorithms, although clock skew compensation can also be seen as intermediate layer between high-level synchronization schemes and timestamp transmission/reception primitives.

Existing Schemes

Simple one-way broadcast synchronization. The simplest high-level synchronization algorithm works only for one-hop synchronization. The idea is that one node acts as a time server, while all other one-hop nodes act as clients. The number of synchronization packets are kept at a minimum, as no further communication is necessary. If a new node joins this one-hop synchronization network, it simply has to wait until enough packets have been received to estimate the server's clock. While this scheme is easy to implement, the synchronization accuracy can still be improved by using another, more sophisticated high-level synchronization scheme.

Round-trip synchronization. NTP [34] is a networking protocol for clock synchronization that is built upon a timestamp transmission/reception primitive. Figure 2.5 shows the working principle, which is based on two nodes: a server node (A) and a client node (B). The algorithm works as described below:

1. Node B requests the current clock of node A at time T_1 using a timestamp transmission primitive.
2. Node A receives the query at T_2, processes it, and replies at T_3.
3. As soon as Node B receives the response, it can calculate the clock offset t_{offset} using the following formula:

$$t_{offset} = \frac{(T_2 - T_1) + (T_3 - T_4)}{2} \tag{2.1}$$

   Under the assumption that the transmission and reception of both messages took the same time.

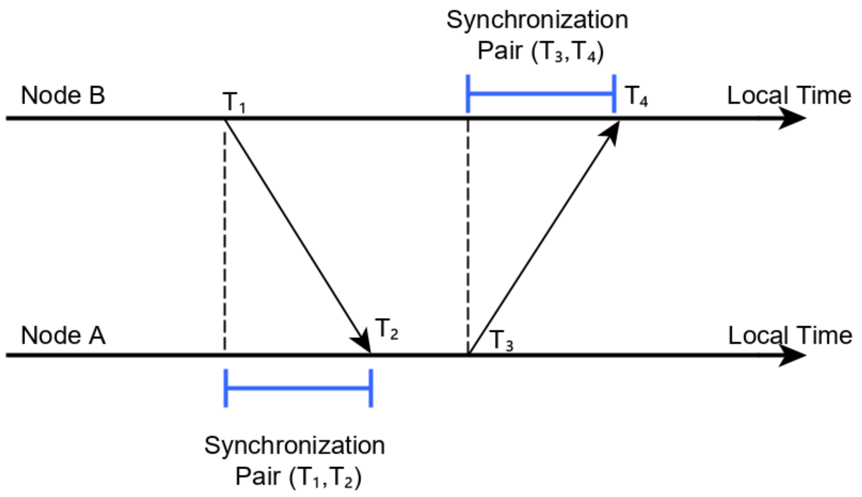


Figure 2.5: Working principle of NTP's clock synchronization algorithm.

RBS. The reference broadcast synchronization scheme [35] and its application in WSNs [36] is fundamentally different from other clock synchronization schemes, as it is based on receiver-receiver clock synchronization. Instead of sending timestamps, a reference node broadcasts a reference beacon that does not contain any timing information to all receivers. Upon reception, the receivers note their local time and then exchange timing information with their neighbouring nodes. The reference clock of the network can then be calculated by comparing these timestamps, even in multi-hop networks.

Prior to perfoming this synchronization, RBS [35] describes ways to choose the reference node, which should periodically send the reference beacons. After successful synchronization, a least squares fit algorithm is used to account for clock skew variations between nodes. The main advantage of RBS is the short critical path, which only includes the transmission and reception delay of the reference node, which is typically in the μs range.

TPSN. The timing-sync protocol for sensor networks (TPSN) [37] is a high level synchronization scheme for WSNs that uses a tree to organize the network topology for multi-hop networks. It consists of two consecutive phases: the level discovery and the synchronization phase. The former is run once upon network deployment and establishes a tree network structure where the reference time server is on level 0, also called the root node. If no distinct server is defined, the role of the time server is periodically switched between different nodes. Once the network topology is constructed, the latter defines pair-wise synchronization between nodes beginning at the root node and traversing the tree. Each synchronization uses a round-trip algorithm to further increase the achievable accuracy.

By using MAC timestamping as a timestamp transmission/reception primitive, TPSN claims to achieve a twice as good precision than RBS [37]. However, it does not define a skew rate estimation algorithm, so the synchronization accuracy mentioned can only be achieved by periodic runs of the synchronization phase.

FTSP. The flooding time synchronization protocol (FTPS) [38] is similar to TPSN but, instead of using pair-wise synchronization, the root node broadcasts its reference time using the MAC timestamping primitive to all reachable client nodes. Upon reception, these nodes will adjust their clocks and act as the new reference node for all the next bunch of reachable client nodes. In this way, a mesh network will be formed, where all nodes will be synchronized to the root node. A big advantage of FTSP is that topology changes are now handled with ease and no new tree structure has to be discovered like in TPSN. In addition to that, FTSP defines methods to periodically choose the best suitable root node, as well as methods for skew rate corrections based on linear regression.

2.3 Cross-Technology Communication

Cross-technology communication (CTC) allows a direct communication between devices with incompatible physical layers. The physical layer of any wireless technology specifies the modulation scheme for sending and receiving data. Due to the varying requirements, these modulation schemes are highly optimized to specific applications and are not interoperable by nature (e.g., a Wi-Fi device cannot directly communicate to a BLE device). With CTC, arbitrary packets can be sent directly between devices with different PHY layers. CTC can be divided into two groups: packet-level CTC and PHY layer emulation.

2.3.1 Packet-level CTC

Packet level CTC can be used to transmit arbitrary data between incompatible physical layers if the following fundamental pre-requisites are fulfilled.

- The used frequency bands overlap with each other. Figure 2.6 shows the fully overlapping channels of BLE and IEEE 802.15.4 in the 2.4 GHz band.
- The current state of the channel can be queried. The IEEE 802.15.4 standard uses a carrier sense multiple access with collision avoidance (CSMA/CA) algorithm which can be seen in Figure 2.7. Albeit BLE does not support CSMA/CA, the most prominent available radio vendors, e.g., Nordic, Texas Instruments, and STMicroelectronics, include

vendor-specific extensions to retrieve the RSSI state information for diagnostic purposes..

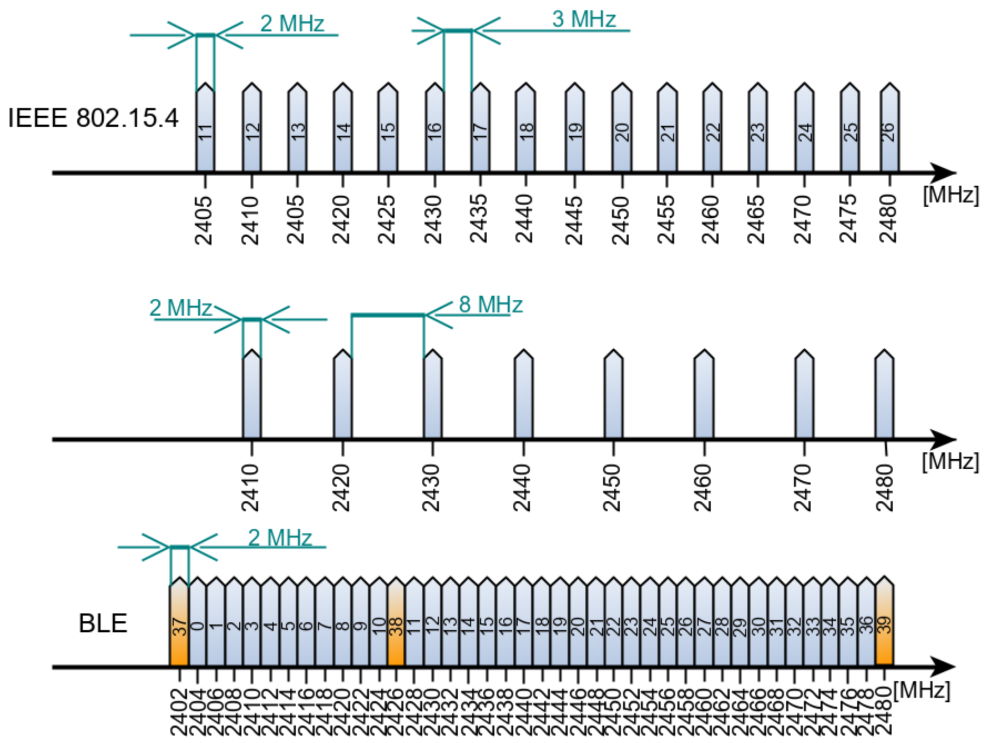


Figure 2.6: Overlapping frequency spectrum of BLE and IEEE 802.15.4.

By sending conventional arbitrary packets on the correct overlapping frequency, the energy on the channel will increase significantly. A radio operating on the same physical layer is hence able to detect and decode the packet on the channel. However, radios with incompatible PHY layers cannot decode the sent packet. Instead, they can sense the intensity and the duration of the received noise by continuously observing the RSSI value i.e., by sampling the RSSI register at high frequency. In packet-level CTC, data can hence be encoded by either sending legitimate data packet, i.e., energy bursts, with different lengths (i.e., encode information in the duration of energy bursts) or by varying the transmission power (i.e., encode information in the intensity of energy bursts). Figure 2.8 shows the CTC transmission

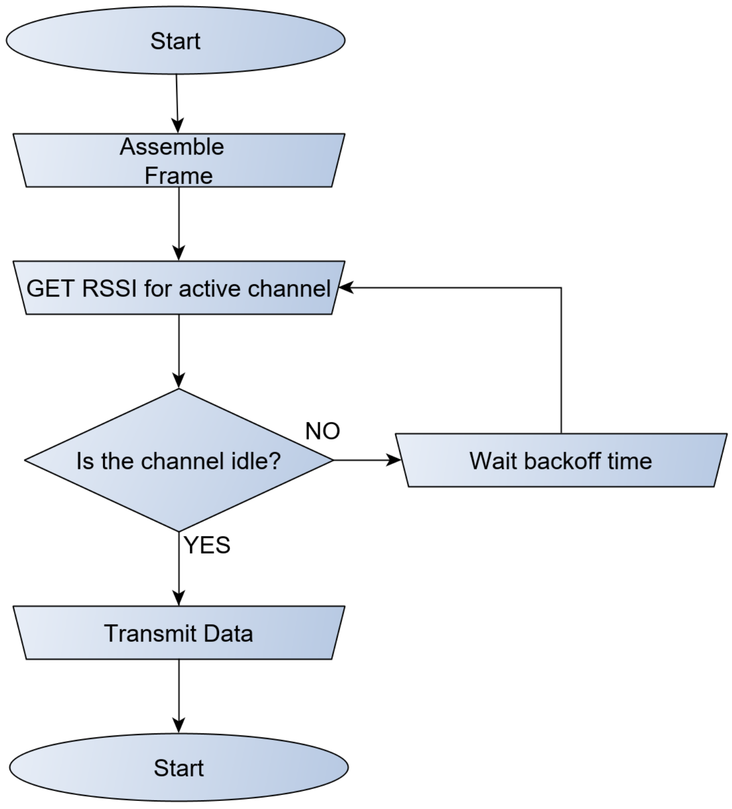


Figure 2.7: CSMA/CA algorithm used in the IEEE 802.15.4 standard.

principle making use of the received signal strength information.

Below, several examples of state-of-the-art packet-level CTC schemes are given:

B^{2}W^2. B^{2}W^2 is a cross-technology communication framework from BLE to Wi-Fi that still allows standard Wi-Fi to Wi-Fi communications. It relies on changing the transmission power level of adjacent BLE packets and embedding a discrete sine wave into these levels. This "discrete amplitude frequency shift keying converter" is used to map symbols to power levels of valid BLE packets in different overlapping channels. In principle, decoding works by obtaining a Wi-Fi channel state information (CSI). One major drawback of this technology is that the power level embedded in this

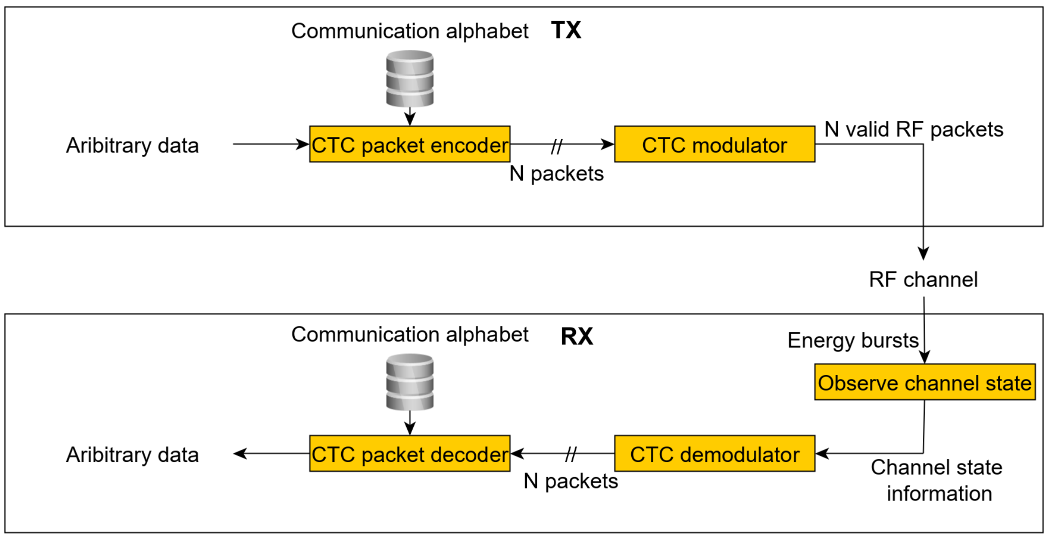


Figure 2.8: Simplified illustration of packet-level CTC transmissions. Arbitrary data will be encoded using a packet-level CTC encoder and a communication alphabet and gets then sent on the shared RF channel. The receiver will continuously sample the channel state information to obtain information about the energy burst length and intensity, which can then be decoded using the same principle.

information can only be read when collisions have occurred during the transmission. In addition to that, only unidirectional transitions from BLE to Wi-Fi are possible.

FreeBee. FreeBee is a cross-technology communication framework between BLE, ZigBee and Wi-Fi [39]. The basic idea is to use Wi-Fi's existing beacon frames to encode data using pulse position modulation (PPM). Due to the re-using of the already existing beacons, no additional communication overhead is created. However, only a very low bandwidth of 30 bpm can be achieved. Furthermore, this approach is very sensitive to noise and only the three advertisement channels of Wi-Fi can be used for transmitting data.

X-Burst. X-Burst is a cross-technology bi-directional communication scheme between BLE and ZigBee devices [20]. It uses precisely-timed energy bursts to transmit information between incompatible physical layers. The arbitrary data is encoded in packet lengths and intervals (gaps) between these energy bursts. It is highly configurable and can be optimized to either achieve a high-reliability or a high transmission rate. It was first implemented on the popular TI CC2650 LaunchPad and Sensortag [40], but has lately been ported to more hardware platforms and also supports broadcasts, as well as Wi-Fi platforms which was showcased in [22]. As X-Burst is a fundamental building block of this thesis, it is described in more detail in Section 2.4.

2.3.2 PHY Emulation CTC

The physical layer describes the raw access scheme to the used wireless channel in the defined frequency band. It therefore defines the modulation scheme, which greatly influences radio characteristics such as transmission speed, reliability, and energy efficiency. Often, two PHY layers with similar characteristics share similar modulation schemes. Several recent works show that this fact can be used to emulate another PHY layer within the sent packet. Figure 2.9 shows the working principle.

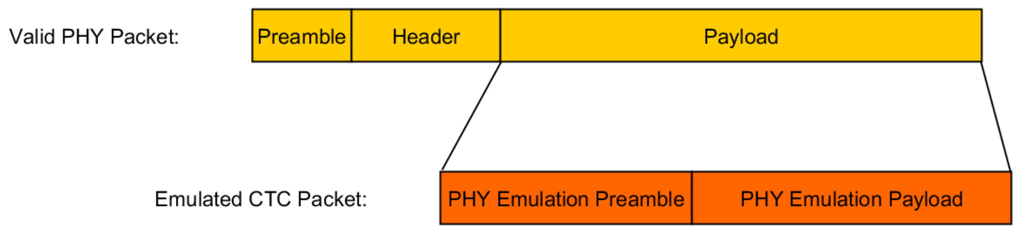


Figure 2.9: Working principle of PHY emulation schemes. An emulated CTC packet is embedded into a valid PHY packet.

The achieved transmission speed can be very high and error tolerant, as long as the modulation schemes are similar. The problem is that this approach cannot be extended easily because it takes a lot of analytical work to calculate the mapping symbols between two PHY layers, as well as to generate the required emulated preamble. If a solution exists and is found, it is very easy to send and receive CTC packets. However, a mapping scheme between two schemes does not necessarily exist. If this is the case, this approach is not applicable. Furthermore, broadcast messages to multiple devices with different PHY layers is not possible, as only one technology can be emulated at a time. For the reasons mentioned above, in this thesis, we will use packet-level CTC as foundation for clock synchronization.

Several examples of state-of-the-art PHY emulation schemes are given below:

BlueBee. BlueBee allows high data rate uni-directional communications between BLE and ZigBee devices [21]. Due to the similarities between the used modulation schemes, the autors found a way to emulate a valid ZigBee frame within in a standard-complaint BLE packet. This allows very high datarates of up to 225kpbs, but only works in one direction. Although in theory, communications from ZigBee to BLE could be possible, reliable decoding of the received frames is not possible, due to the nature of the used modulation schemes.

WEBee. WEBee achieves uni-directional Wi-Fi to ZigBee transmissions by using PHY layer emulation even in noisy environments [41]. Additionally, WEBee can emulate up to two BLE packets within one Wi-Fi frame, thus resulting in a higher spectrum efficiency. Due to the nature of PHY layer emulation, the frame reception ratio for Wi-Fi to ZigBee transmission is about 50%, which has to be compensated for in the upper layers. The authors claim that enhanced versions of WEBee allow bi-directional communication, which is currently under development.

2.4 X-Burst

X-Burst is a cross-technology, bi-directional, packet-level communication framework between BLE and IEEE 802.15.4 devices [20]. It is designed to be non-technology specific, which is shown by the support of a Wi-Fi radio. Hence X-Burst allows bi-directional communication between Wi-Fi, BLE and IEEE 802.15.4 devices simultaneously, including support for broadcast transmissions [22]. Support for Wi-Fi was not available at the beginning of this thesis and thus is not discussed furthermore. X-Burst uses precisely-timed energy bursts to transmit information between incompatible physical layers. These energy bursts are generated by sending legitimate data packets of variable length. Thereby, the information is encoded in the duration of packets and/or in the intervals in-between. Figure 2.10 shows the working principle of X-Burst.

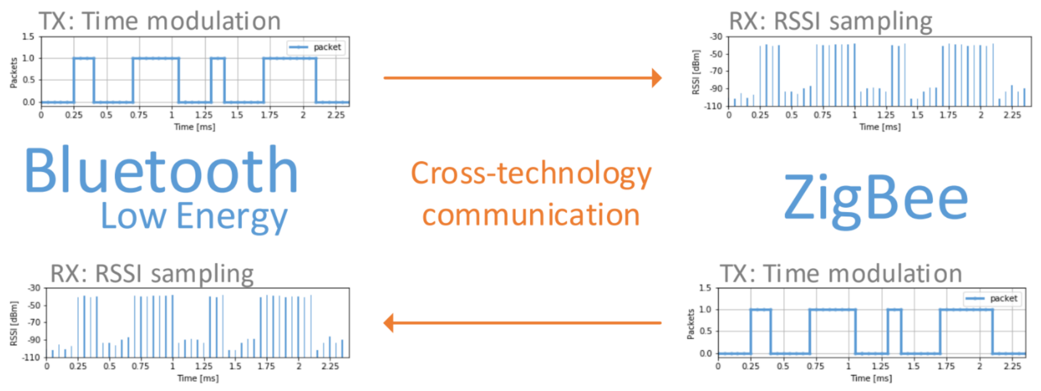


Figure 2.10: Working principle of X-Burst [20].

In order to allow both BLE and IEEE 802.15.4 devices to transmit and receive energy bursts, valid RF packets of the respective physical layer are used. The duration of these energy bursts is therefore discrete and has certain limitations defined in the PHY layer. In addition to that, user policies can be applied to either focus on high data rates or high reliability. All these considerations define the communication alphabet that maps arbitrary data to X-Burst symbols. Table 2.1 shows a sample communication alphabet with the packet length for both BLE and IEEE 802.15.4 using a hexadecimal encoding.

The result of this mapping mechanism are “bursts” and “gaps” that can then be sent on the RF channel. Any PHY-compatible radio can receive this stream of information, where each burst (i.e, transmitted packet) would have variable length. Radios with different PHY layers, however, can receive the CTC encoded data by sampling the RSS at high frequency. Figure 2.11 shows an encoded packet on the RF channel using a hexadecimal mapping, the RSSI sampling intervals, and the received reconstructed packets. Once the packet is fully received, the same mapping scheme will be applied again to recalculate encoded data / information. The recorded energy burst duration can vary because of the slow sampling speed or because of external interference.

Value	Energy Burst Duration [μs]	Payload Bytes BLE	Payload Bytes IEEE 802.15.4
0x0	192	14	0
0x1	224	18	1
0x2	256	22	2
0x3	288	26	3
0x4	320	30	4
0x5	352	34	5
0x6	384	38	6
0x7	416	42	7
0x8	448	46	8
0x9	480	50	9
0xA	512	54	10
0xB	544	58	11
0xC	576	62	12
0xD	608	66	13
0xE	640	70	14
0xF	672	74	15

Table 2.1: X-Burst mapping scheme using a common alphabet and hexadecimal encoding.

One major challenge that X-Burst has successfully mastered is accurately measuring the length of energy bursts, even when the RSSI value is averaged with a moving window, according to the standard (IEEE 802.15.4). A sampling strategy for the reception of X-Burst transmissions on these radios has been developed and successfully evaluated. Instead of using only one threshold to detect the presence of an energy burst, several configurable thresholds are used. In addition to that, during the preamble, X-Burst compares the measured averaged durations to the nominal energy burst durations and calculates a compensation offset value. This offset is then used for the payload of the CTC packet.

X-Burst is implemented using a highly-modular architecture which reduces complexity of the CTC implementation, maximises code reuse and increases the portability to other hardware platforms [20], which is illustrated in Figure 2.12. The hardware abstraction layer defines a minimal set of functions which need to be implemented in order for X-Burst to work, such as `sample_RSS` or `generate_burst`. Building on top of that layer are the encoding and decoding layers, which manage X-Burst transmissions and receptions respectively. At this layer, the durations are known but not the

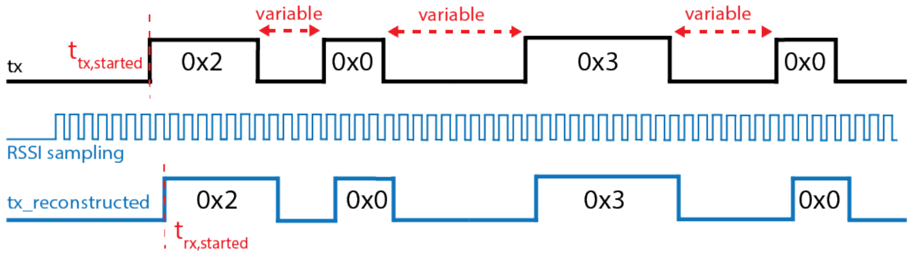


Figure 2.11: Example transmission of a small packet using hexadecimal encoding with burst only modulation (no data is encoded in gaps between bursts).

actual data. With the use of the chosen coding scheme and therefore the chosen communication alphabet, durations can be converted to data and vice-versa. X-Burst transmissions are very versatile, and can be broadcasts, unicasts or multicasts, can include a checksum and/or the payload length and other features defined in the header. The frame management layer handles these flags in the header and communicates with the application.

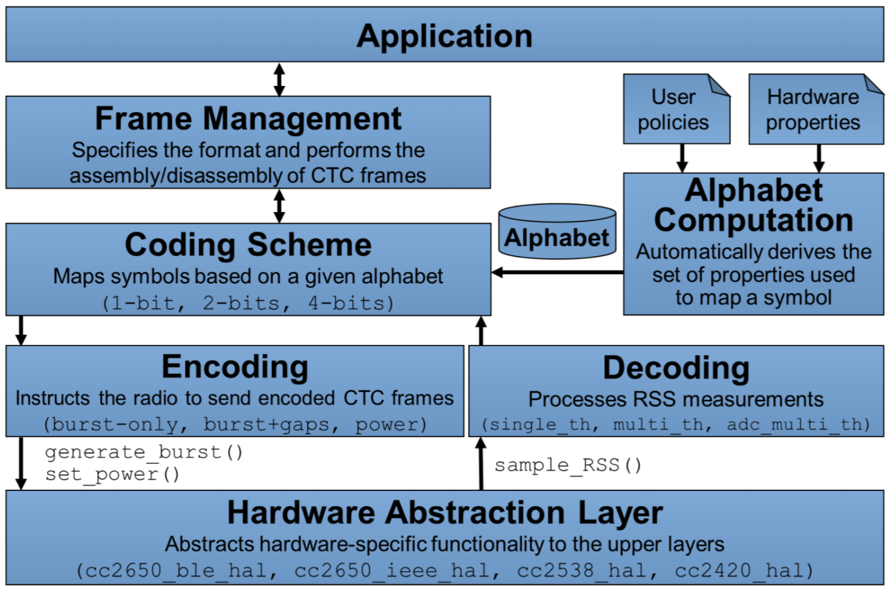

Application
Frame Management
Specifies the format and performs the assembly/disassembly of CTC frames
User policies
Hardware properties
Alphabet Computation
Automatically derives the set of properties used to map a symbol
Alphabet
Coding Scheme
Maps symbols based on a given alphabet
(1-bit, 2-bits, 4-bits)
Encoding
Instructs the radio to send encoded CTC frames
(burst-only, burst+gaps, power)
Decoding
Processes RSS measurements
(single_th, multi_th, adc_multi_th)
generate_burst()
set_power()
sample_RSS()
Hardware Abstraction Layer
Abstracts hardware-specific functionality to the upper layers
(cc2650_ble_hal, cc2650_ieee_hal, cc2538_hal, cc2420_hal)

Figure 2.12: X-Bursts modular architecture, taken from [42].

2.5 Related Work

This section gives an overview about the most important works in the field of cross technology coexistence and cross-technology clock synchronization. The first three Sections (2.5.1 - 2.5.3) focus on cross technology coexistence, where time synchronization is a central aspect. All three works include an evaluation in terms of transmission errors, but no evaluation of the synchronization accuracy. Section 2.5.4 presents Crocs, to date, the only other existing cross-technology clock synchronization scheme.

2.5.1 Coexistence between IEEE 802.15.4 and IEEE 802.11 through cross-technology signaling

The work of Bauwens. et. al. deals with the coexistence problem of IEEE 802.15.4 and IEEE 802.11 nodes [43], which conventionally results in performance degradations. Their vision is that devices should be able to negotiate medium access with each other, instead of implementing a contention based access. They use a novel cross-technology time division multiple access (TDMA) scheme, which works due to synchronization using an energy pattern beacon. The working principle can be seen in Figure 2.13. A periodic IEEE 802.11 beacon is sent from the central node. All other nodes can receive this pattern by sampling the current energy on the shared radio channel and can use this information to time their transmissions according to the predefined schedule. Thereby, their goal is to minimize collisions, but not to maximize synchronization accuracy, which means that nodes must only listen for beacons if their clock drift is above a certain threshold. To show the feasibility of the proposed TDMA scheme, it was successfully evaluated in a large scale testbed.

A demo regarding the developed TDMA scheme [44] showcases this novel solution. The impact of parameters such as distance to beacon transmitter, CCA threshold, and interference were evaluated. The conclusion of this demo is that a cross-technology TDMA scheme is a viable option to mitigate cross-technology interference.

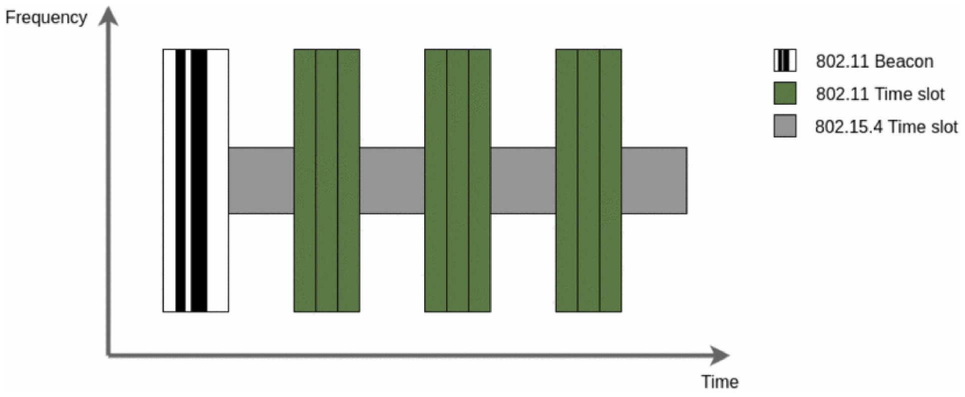


Figure 2.13: Working principle of the cross-technology TDMA scheme proposed in [43].

2.5.2 Cross-technology wireless experimentation: Improving IEEE 802.11 and 802.15.4e coexistence

The authors of [45] created a novel wireless experimentation framework called WiSHFUL that facilitates the prototyping and experimental validation of innovative solutions for heterogeneous wireless networks. The proposed framework aims to separate the definition of logical behaviour in wireless networks and the underlying device capabilities. One application of this framework is cross-technology interference mitigation between IEEE 802.11 and IEEE 802.15.4e nodes, which is shown by implementing an on-the-fly configurable TDMA scheme (Figure 2.14). The authors presented their solution in an interactive demonstration which showed the effectiveness of cross-technology interference mitigation and allowed the users to better understand the WiSHFUL framework. The relative synchronization accuracy of the IEEE 802.11 and IEEE 802.15.4e nodes was not evaluated.

2.5.3 Exploiting Programmable Architectures for Wi-Fi/ZigBee Inter-Technology Cooperation

The authors of [46] created a cross-technology TDMA scheme, similar to [43] and [45] by utilizing programmable platform-agnostic architectures, namely Wireless MAC processor (WMP) [47] and SnapMAC [48]. These

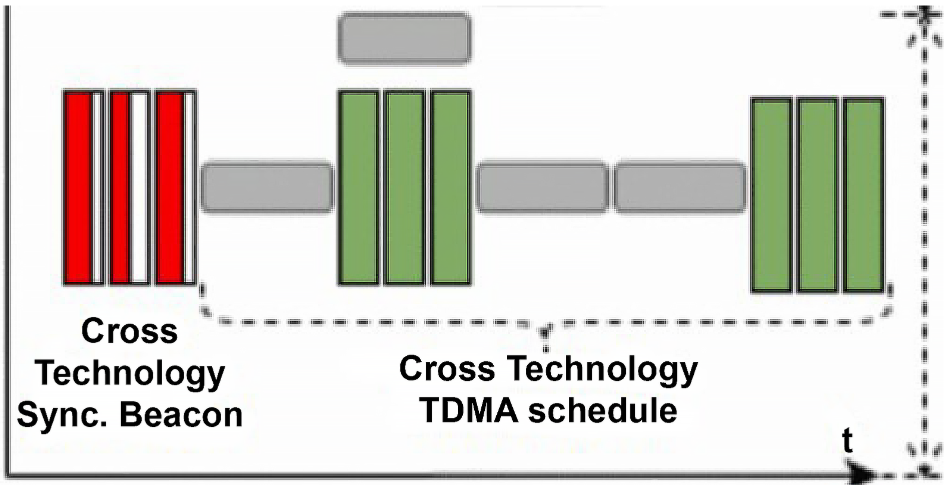


Figure 2.14: Working principle of the cross-technology TDMA scheme proposed in [45].

architectures allow to define a simple TDMA scheme in a hardware-agnostic, cross-platform manner, as shown in Figure 2.15.

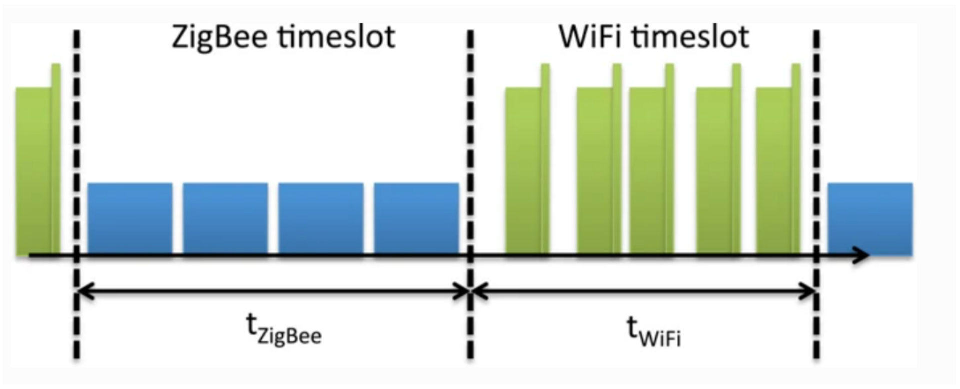


Figure 2.15: Working principle of the cross-technology TDMA scheme proposed in [46].

In their evaluation, the authors show that by using the TDMA protocol, this lost time is recovered, and the channel is used much more efficiently without negatively effecting the Wi-Fi throughput. The relative synchronization accuracy of the Wi-Fi and ZigBee nodes was not evaluated.

2.5.4 Crocs

Crocs is a cross-technology clock synchronization scheme for synchronizing ZigBee to Wi-Fi networks. CTC is achieved by using the channel state information sensing capabilities of the ZigBee nodes. The low software sampling speed of these RSSI values makes the use of a synchronization mechanism necessary. Crocs achieves that by sending Wi-Fi packets with data encoded in the timespans between adjacent transmissions. Figure 2.16 shows the high-level operating principle of Crocs, which is divided into two phases: the alignment phase and the timestamp transmission phase.

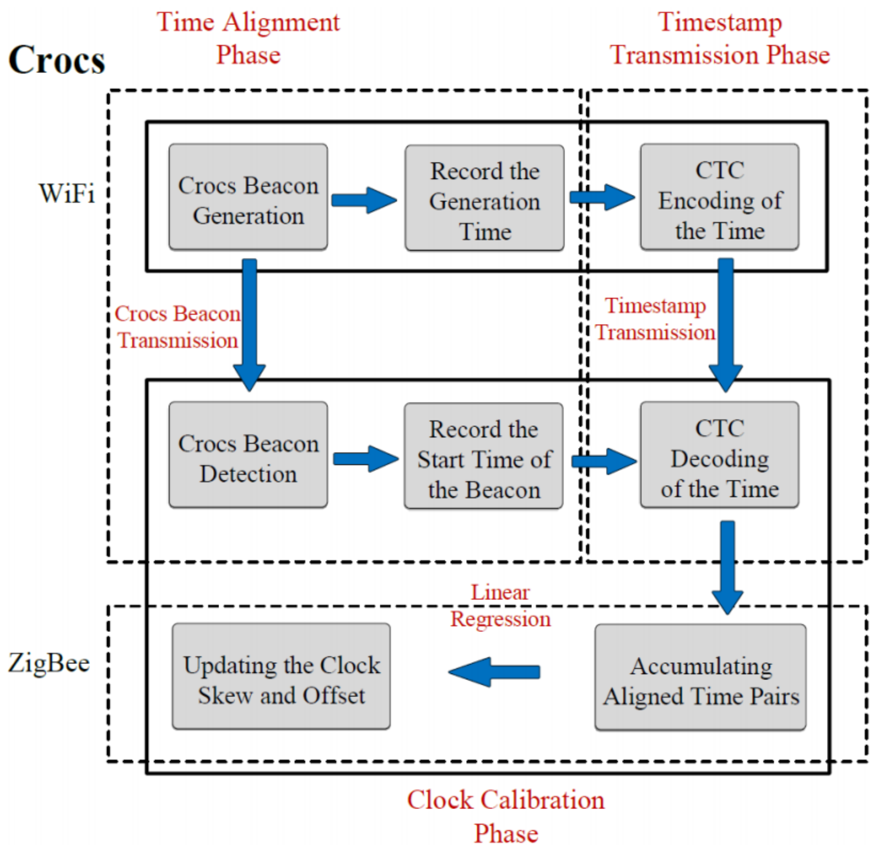
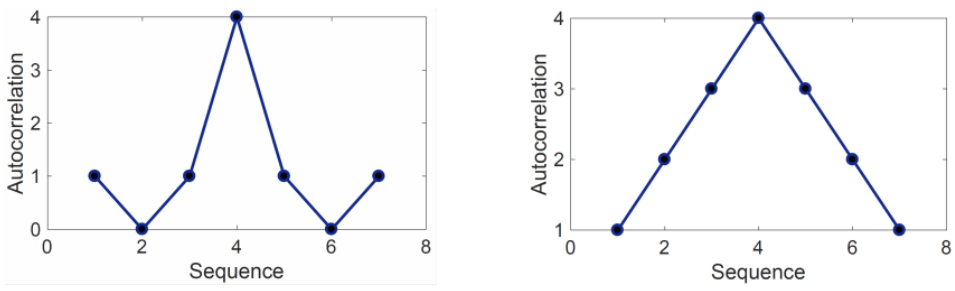


Figure 2.16: The high-level operating principle of Crocs [49].

Clock alignment phase. A specially encoded sequence called the Barker sequence will be sent to the ZigBee node by transmitting Wi-Fi packets with data encoded in the time between adjacent transmissions. Upon reception, the receiving node makes use of the autocorrelation function, which correlates the received signal to itself. The exact same calculation is done at the transmitter. The used Barker code possesses the ideal autocorrelation property, which results in a peak at a specific point in time (that will be the same at both the transmitting and the receiving node). This peak will be used at the transmitter and at the receiver to sample the current local clock. Figure 2.17 shows the autocorrelation of the used Barker sequences. The peaks in this sequence are used to obtain synchronization between a Wi-Fi and a ZigBee node.



(a) Autocorrelation for the Barker code $(+1,+1,+1,-1)$ (b) Autocorrelation for the sequence $(+1,+1,+1,-1)$

Figure 2.17: Barker Sequences used in Crocs [49].

Timestamp transmission phase. The stored timestamps will be transmitted afterwards in a non time-critical CTC transmission. Upon reception, the receiving node now has a synchronization pair consisting of the transmitting Wi-Fi nodes timestamp T_1 and the receiving ZigBee nodes local T_2 timestamp. [49] defines two different modulation schemes, which can both be used to transmit the created timestamp of the Wi-Fi transmitter to the ZigBee client. When the packet is successfully transmitted, the ZigBee node has now both timestamps and can correct its clock, as well as estimate the skew rate.

In the evaluation, Crocs achieves a synchronization accuracy of up to 1ms. To achieve that, a USRP system was used to generate the required Wi-Fi synchronization sequence and to transmit the timestamp using CTC. To conclude, Crocs has certain drawbacks which do not make the system usable:

- no off-the-shelf hardware used;
- uni-directional synchronization scheme only;
- the achievable accuracy of 1ms is not enough for IoT applications.

2.6 Employed Hardware

To demonstrate the generic design of X-Sync, as well as to allow future portability, we implemented the scheme on several off-the-shelf IoT development platforms. The employed hardware platforms, which are listed below, are largely diverse, and range from low clock speeds (8MHz) and only IEEE 802.15.4 support to high speed (40MHz), dual radios (BLE and IEEE 802.15.4). Hereby, the available memory on each platform is given in Table 2.2.

Node	available ROM [kB]	available RAM [kB]
CC2650 (BLE)	128	28
Firefly (IEEE 802.15.4)	512	32
TelosB (IEEE 802.15.4)	48	10

Table 2.2: Summary of the available memory for the used off-the-shelf IoT development platforms.

2.6.1 TI CC2650 Launchpad and Sensortag

Both the TI CC2650 Launchpad and the Sensortag platform are based on the Texas Instruments CC2650 radio chip, which offers a low-cost and multi-standard package supporting both BLE 4.2 and IEEE 802.15.4 in a single package [40]. The chip consists of two integrated CPUs: the ARM Cortex-M3 main core and a separate Cortex-M0 core for the radio communication. The former provides a 48 MHz clock and several low-power modes, while the latter handles time-critical radio communications. Communication between these two is handled by the Communication Packet Engine (CPE) subsystem, which provides an easy way to send and receive packets. Figure 2.18 shows a block diagram of the RF subsystem which is fully supported in Contiki and Contiki-NG.

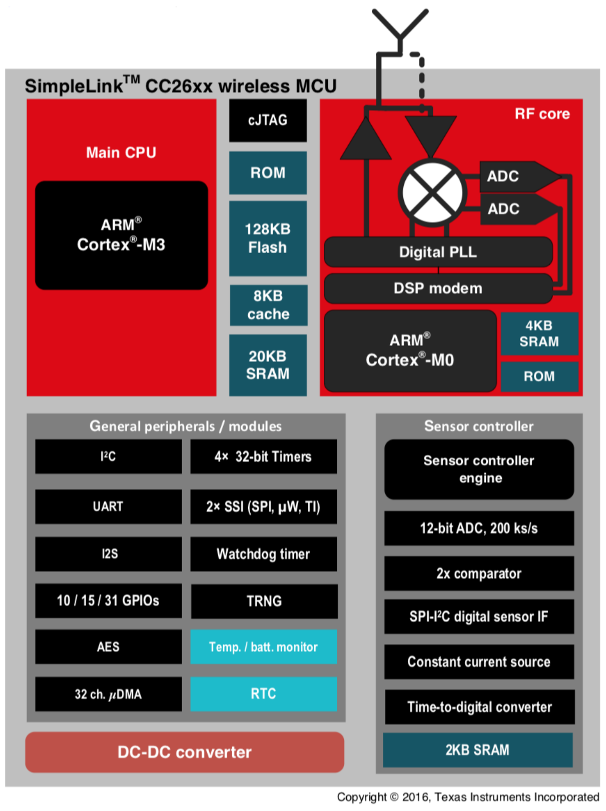


Figure 2.18: CC2650 functional block diagram [40].

2.6.2 TelosB Mote

The TelosB Mote [50] is an ultra low-power IEEE 802.15.4 compliant wireless sensor module. It is based on the MSP430 microcontroller paired with an IEEE 802.15.4 compatible transceiver (TI CC2420 [51]). Figure 2.19 shows the functional block diagram. The CPU is clocked at up to 8 MHz and offers an extended random access memory of 10 kB that is sufficient to run a small operating system. This development platform is officially supported by both Contiki and Contiki-NG. The Texas Instruments CC2420 transceiver is a single-chip 2.4 GHz IEEE 802.15.4-compliant and ZigBee-ready RF transceiver, which is interconnected to the main CPU using a 4 wire SPI interface. Additional features include 1 MB external flash space as well as light, temperature, and humidity sensors. Due to the popularity and the open-source hardware design, many clones are existing today (e.g., Advanticsys MTM-CM5000 [52] and the Moteiv Tmote Sky [24]). In this thesis, the Advanticsys MTM-CM5000 clone is used.

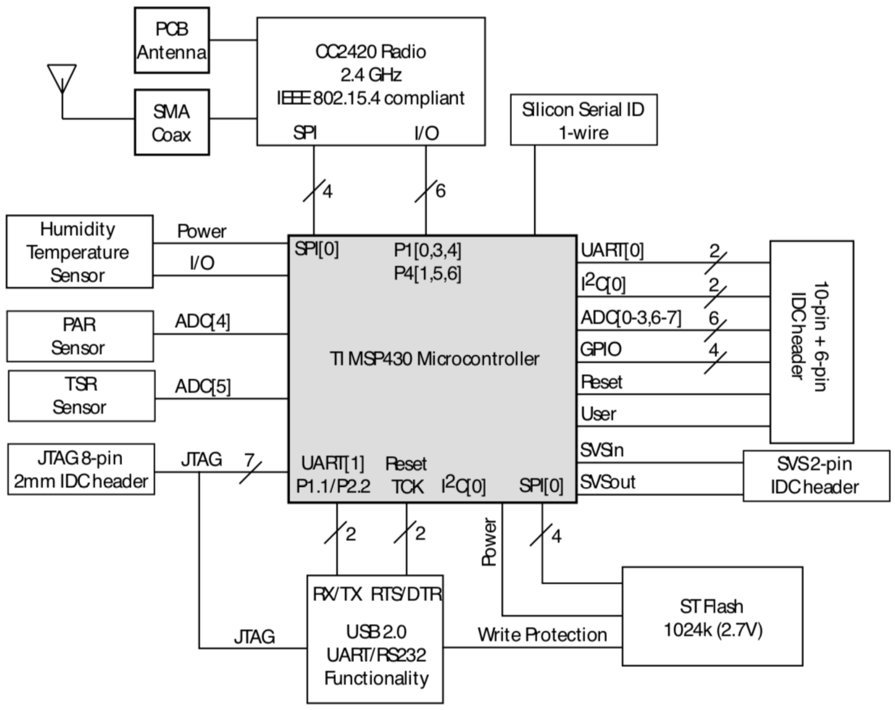


Figure 2.19: Functional block diagram of the TelosB open-source hardware design, its components, and buses [24].

2.6.3 Zolertia Firefly

The Zolertia Firefly development platform [25] was designed for 2.4 GHz as well as 863-950 MHz IEEE 802.15.4 applications and is supported in both Contiki and Contiki-NG. It features a CC2538 ARM Cortex-M3 microcontroller [53] clocked at 32 MHz with an on-chip 2.4 GHz IEEE 802.15.4-compliant RF transceiver. In addition to that, a CC1200 low-power, high performance RF transceiver [54] is externally connected via SPI. This radio allows to use a number of different frequency bands, including the ISM/SRD bands 169, 433, 868, 915 and 920 MHz.

2.7 Employed Software

In order to show the portability as well as the generality of our solution, we implemented X-Sync on different hardware platforms. Operating systems (OS) provide hardware-independent abstraction mechanisms to allow sharing a common codebase between different development platforms. Due to very strict timing requirements within low-level timestamp transmission and reception, hardware-specific code has to be developed as well. However, abstraction into more generic operating system layers is always preferable. Due to its concurrency support, its high popularity, as well as its active development, Contiki-NG was chosen as the operating system.

Contiki-NG describes itself as an OS for the next generation of IoT devices. Contiki-NG is open source, supports several platforms, and focuses on (secure and reliable) low-power wireless communication [55]. It is a fork of the original Contiki OS and will most likely become its up-to-date successor. Hereby, Contiki-NG focuses on standard-based IPv6 communications, modern IoT platforms (mostly 32-bit MCUs), and right on documentation. To achieve these goals, it follows a agile development process with periodic releases. By using this codebase, it is possible to write platform-agnostic code with support for several higher layer protocols such as IPv4, IPv6, UDP, and TCP. Platform-specific code is isolated from the main system and can also be extended and modified. Like the original Contiki, Contiki-NG has built-in support for protothreads [56], which are a low-level mechanism for concurrent programming. Protothreads are stackless, lightweight, and non-preemptable threads ensuring that a running program block will be executed in the correct order, while native, platform-dependent interrupts are still possible.

3 X-Sync: Design

The first step in designing a CTCS scheme is define the requirements (Section 3.1). Thereafter, we present a system overview in Section 3.2, where the design rationale and working principle of our X-Sync CTCS scheme is discussed in detail. Afterwords Sections 3.3, 3.4, and 3.5 discuss in more detail how X-Sync tackles a number of challenges in fulfilling the requirements listed in Section 3.1.

3.1 Requirements

The requirements listed below are set by hardware and software limitations by common applications, as well as by the state-of-the-art in the field of clock synchronization.

The relative synchronization error between two synchronized nodes has to be below 2.3μs under ideal conditions. The relative error between two synchronized sensor nodes is an essential factor for the validity and integrity of distributed measurements. Depending on the type of application, this error should be as low as possible, especially for real-time, industrial environments. State-of-the-art synchronization protocols for homogeneous wireless networks achieve synchronization accuracies of up to 1.5μs in a one-hop scenario [57]. Petros et al. [58] conclude that for certain specialised applications such as power lines and grids, an absolute accuracy of up to 1μs is needed. To put this number into perspective, Figure 3.1 shows the accumulated timing error from a commercial off-the-shelf crystal oscillator, which is embedded into popular IoT platforms. The authors of [58] further claim that the precision time synchronization protocol (PTP) in an IEEE 802.11 network can achieve a maximum accuracy of 2.3μs under ideal conditions. All other compared synchronization schemes in [58] are either hardware based or use additional hardware such as time references and are therefore not usable for this thesis. Existing works in cross-technology clock synchronization achieve millisecond accuracy [49], which simply is not sufficient for most applications, e.g., in industrial IoT applications [59]. For the aforementioned reasons, one goal of this thesis is to sustain a synchronization accuracy of

2.3μs under ideal conditions, which corresponds to the maximum accuracy of PTP. This would mean that, although a cross-technology synchronization scheme is used, no accuracy is lost compared to a conventional solution for homogeneous networks.

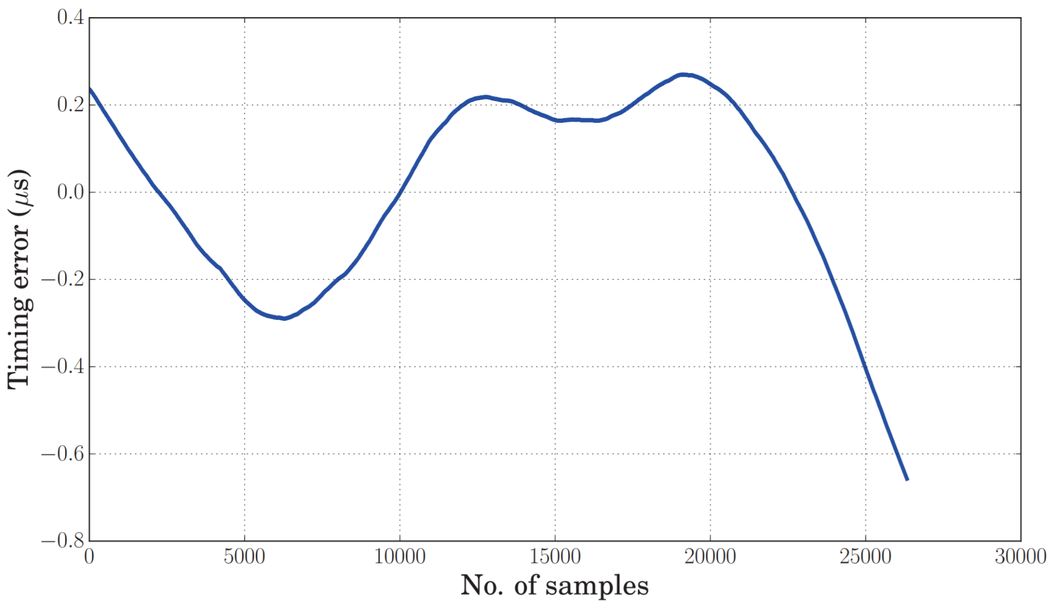


Figure 3.1: Accumulated timing error from a commercial-off-the-shelf crystal oscillator. Image taken from [60].

The synchronization between two nodes should be reliable. Packet-level cross-technology communication works by continuously sampling the energy level on the channel. Any noise on the channel will hereby influence the timing of the transmitted energy bursts and has direct impact on the synchronization accuracy. Thus, disturbances are more likely than in homogeneous systems and have a bigger impact on data integrity. X-Sync has to be reliable not only in low-noise laboratory experiments, but also in real-world conditions.

Low memory footprint. If clock synchronization is needed for a specific application, there should still be enough memory available on the node for the application logic. The proposed solution should be configurable in terms of memory consumption, albeit this could lower the sustained synchronization accuracy. Despite these limitations, X-Sync should run on older, memory limited development platforms, such as the TelosB.

High energy efficiency. Wireless sensor nodes are often used in power-constrained environments. Therefore, energy efficiency is significant, as the available energy is limited and replacing batteries is expensive and often not feasible in remote locations. X-Sync should hence allow to trade energy efficiency against synchronization accuracy, which allows configuring X-Sync to the requirements of the application and device.

Generality. Nodes with lower processing capabilities should be supported by X-Sync seamlessly. However, not all hardware platforms have access to accurate, high frequency timers. X-Sync should run on these platforms as well, albeit with a reduced accuracy due to these limitations. Furthermore, X-Sync should be portable to other hardware platforms, other operating systems and other wireless technologies which allow RSS sampling.

3.2 Overview

In this section, we introduce X-Sync, a novel cross-technology clock synchronization scheme that allows to seamlessly exchange timestamps between IEEE 802.15.4 and BLE devices with a microsecond-level accuracy. It achieves that by extending the packet-level CTC scheme X-Burst in a way to allow a high-accuracy clock synchronization. Figure 3.2 gives an overview of the X-Sync CTCS scheme, which we divide into three sections: transmission, reception, and validation/clock correction.

Transmitting. Device A starts by sending a CTC preamble on the shared cross-technology radio channel. Immediately, the current clock at the transmitter is sampled and stored in T_1, which will get transmitted at a later stage. Next, the transmitter sends the synchronization preamble which is needed for clock compensation at the receiver. Lastly, the header, a payload (if any) and the formerly stored timestamp T_1 are added to the CTC transmission. The whole transmission and especially the transmission of the synchronization preamble rely on having accurate timing behaviour, as discussed in detail in Section 3.3, and primarily depends on the following:

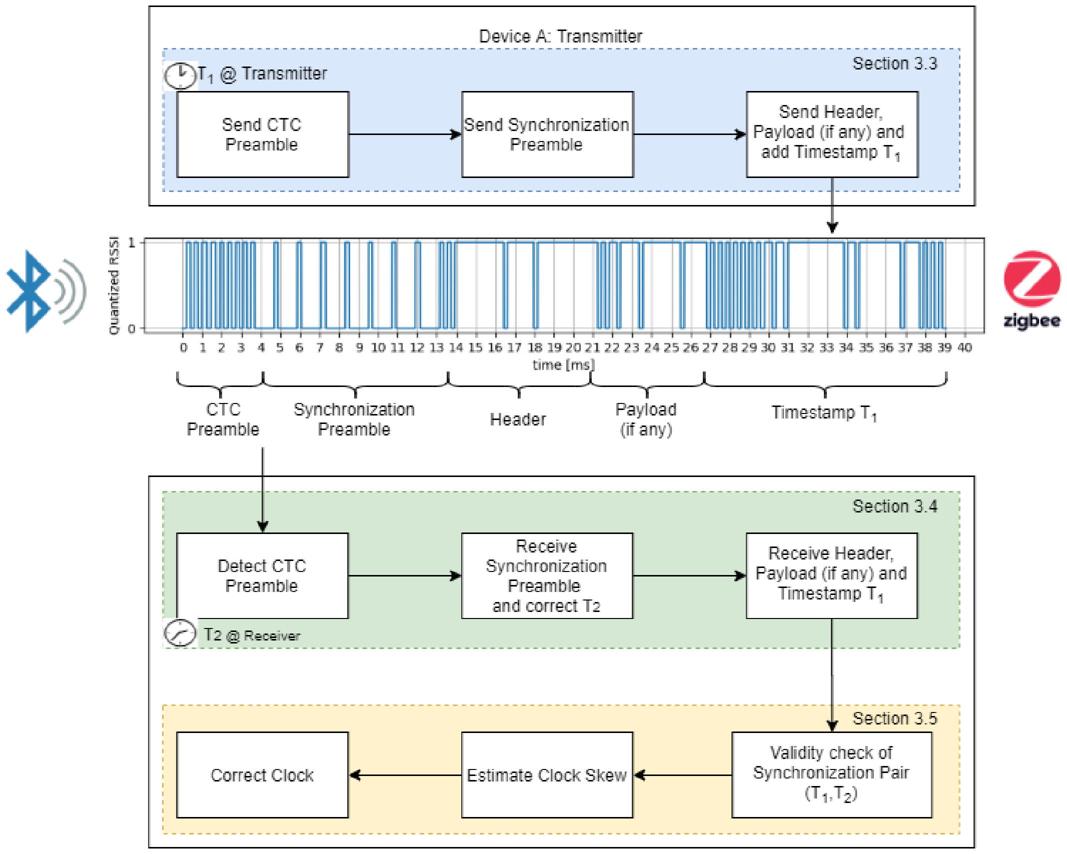


Figure 3.2: Overview of the X-Sync CTCS scheme, divided into a transmitter (device A) and a receiver (device B) section with the shared cross-technology channel illustrated in between. Device A begins by sending the CTC preamble, followed by a synchronization preamble and the actual timestamp T_1. Device B detects CTC preamble on the shared channel, synchronizes upon the synchronization preamble and finally receives the timestamp T_1. At this stage, device B knows both timestamps T_1 and T_2, checks them for validity and uses them to correct its local clock. The exact moment when T_1 and T_2 are sampled, at the transmitter and the receiver respectively, are indicated explicitly.

- The radio state machine (i.e., a limited set of states handling the transmission and reception of radio packets) introduces several deterministic and non-deterministic delays during the transmission of radio packets, which limits the achievable accuracy.
- During the transmission of multiple data packets, the CPU must not

be interrupted by other tasks (e.g., interrupts). Inaccurate energy burst timings greatly decrease the synchronization accuracy.

Receiving. With the detection of the CTC preamble, device B immediately samples the reception time T_2. This timestamp will be corrected and compensated for delays introduced due to the use of CTC by using the synchronization preamble. The packet is then fully received, and the timestamps T_1 and T_2 form a clock synchronization pair. This process relies on having an accurate reception timing, as discussed in more detail in Section 3.4 and depends on the following:

- The RSS sampling frequency is slow (e.g., 40kHz on some platforms) and limited by the underlying hardware. Therefore, the detection accuracy of the start of an energy burst is limited and has a direct impact on the synchronization accuracy.
- The IEEE 802.15.4 standard is specifying a moving average RSS filter that has to be implemented on the radio state machine. Thus, the actual RSS value on the RF channel cannot be obtained. This greatly reduces the achievable sampling precision.
- The reception process must not be disturbed by any other event happening on the microcontroller, such as interrupts. These events could lead to wrong energy burst timings, which decreases the synchronization accuracy.

Validation and correction. Once such a clock synchronization pair (T_1, T_2) is available at device B, it is further processed and checked for validity by the use of RANSAC. RANSAC is an iterative method to detect outliers within a set of observed data. Due to the energy sampling nature of packet-level CTC, transmissions are not equally reliable as traditional, homogeneous networks. RF interference can hereby manifest in two ways:

- RF interference can lead to wrongly decoded symbols and therefore wrong timestamps. Such transmission errors need to be detected and must not be used for the synchronization process.
- RF interference can lead to wrong reception timings, which strongly affects the achieved synchronization accuracy. Such `outliers` must be detected and filtered during skew rate estimation.

Once the received clock synchronization pair is checked for validity, clock skew estimation is needed to reduce energy consumption and increase the long-term synchronization accuracy. Finally, the local clock of the receiving node can be corrected. This whole process is discussed in more detail in Section 3.5.

3.3 Precise Transmission Timing of CTC Messages

The averaging delay $\tau_{averaging}$ compensation methods mentioned in Section 3.4 rely on having an accurate transmission timing as a substitute for high-frequency sampling. X-Sync makes use of high-precision clocks (Section 3.3.1), radio queues (Section 3.3.2) and interrupt priority masking (Section 3.3.3) to achieve that. X-Sync transmissions also incorporate MAC timestamping (Section 3.3.4) to compensate the send and access times on the transmitting node.

3.3.1 High-Precision Clocks

A pre-requisite of a high-precision clock synchronization scheme is to have a stable, high frequency clock source connected to a timer. Hereby, properties of such clocks vary greatly among the available hardware. Each available timer was therefore evaluated and compared in terms of stability and frequency. Furthermore, the free running timer is often limited to either 16 or 32 bits, which is not enough to keep track of the current time for a long period. Thus, these timers had to be extended by counting the overflows and extending the maximum count.

3.3.2 Radio Queues

Radio queues are a hardware feature of IEEE 802.15.4 and BLE radios that allow the user to build a queue consisting of radio commands. These commands include instructions for packet transmission and waiting for a specified duration. X-Sync uses this mechanism to enqueue packets (energy bursts) and wait commands (gaps) in between these packets. Once such a structure is built, the radio state machine will process these commands one by one without being interrupted by other tasks happening on the microcontroller. Precise timing is therefore guaranteed. Figure 3.3 illustrates how energy bursts and gaps are transmitted conventionally. First, the current packet is enqueued and will be processed whenever the radio state machine is ready. After an successful transmission, the microcontroller waits for the amount of time specified for the gap before repeating the whole process. The transmission timing can hereby be influenced by the following events:

- The radio state machine is not always in the same state when the microcontroller enqueues and transmits an energy burst. This introduces an uncertainty.
- The radio state machine has to tell the microcontroller once the transmission is finished: depending on how this is done internally, it can introduce lags.
- During the busy wait state (for creating the gaps), the microcontroller must not be interrupted. If so, the gap timespan can be off.

Figure 3.4 illustrates how the same behaviour can be achieved with exact transmission timings. A radio queue containing energy bursts and gaps is built. This queue can then be processed by the radio state machine one by one, without the need of any microcontroller interaction and with precise timings.

X-Sync has to be versatile to allow both ways of transmission, as some platforms may not have radio queues or similar mechanisms. This will be described in detail in Chapter 5.

3.3.3 Interrupt Priority Masking

Interrupt priority masking is an important hardware feature for radios that do not allow radio queues. Priorities can be assigned to interrupts that can then be masked by setting a priority threshold. Interrupts with a lower priority than the threshold will be masked and, therefore, ignored.

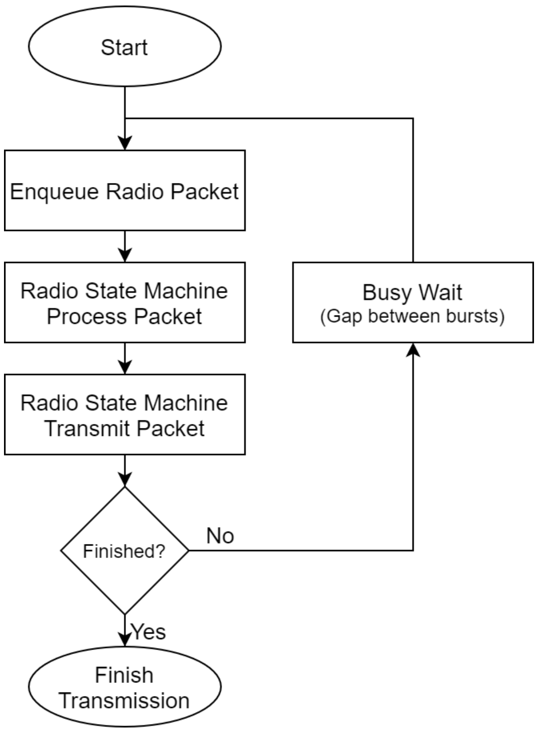

Start
Enqueue Radio Packet
Radio State Machine Process Packet
Radio State Machine Transmit Packet
Finished?
No
Yes
Busy Wait (Gap between bursts)
Finish Transmission

Figure 3.3: Packet-level transmission without the use of radio queues. Energy bursts will be enqueued, eventually processed by the radio state machine and transmitted. The microcontroller is then responsible for waiting the amount of time specified for the gap. This process is repeated until the CTC message is transmitted.

3.3.4 MAC Timestamp Transmission

MAC time-stamping provides a way to compensate delays introduced by the transmitting process (send time, access time). It is based on appending data on the MAC layer in the latest possible moment before the transmission starts. Whenever data is going to be transmitted, it is pushed to the radio core's queue and is stored until the state machine of the radio is ready. This introduces some jitter, as it is not deterministic when the packet is sent. The following steps can be taken to overcome this problem:

1. Register for an interrupt / event when the data is actually sent or n bytes are written;

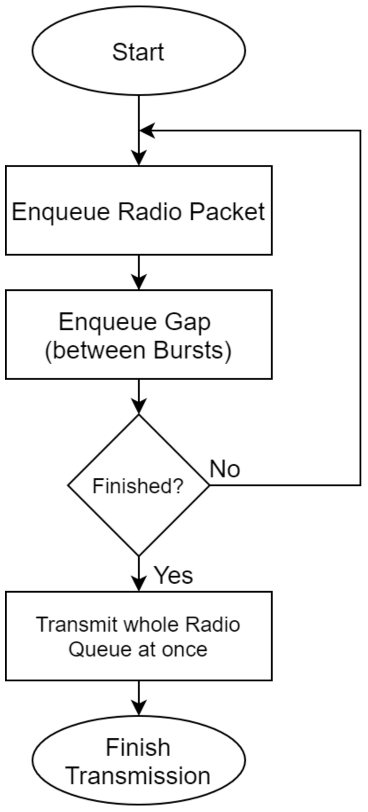

Start
Enqueue Radio Packet
Enqueue Gap (between Bursts)
Finished?
No
Yes
Transmit whole Radio Queue at once
Finish Transmission

Figure 3.4: Packet-level transmission with the use of radio queues. All energy bursts and gaps will be enqueued using the mechanisms provided by the radio. Once all data is processed, the transmission of the whole queue will be triggered. Thereby, the radio state machine is managing the precise burst and gap timings, instead of the microcontroller.

2. Reserve some space for the synchronization timestamp in the X-Sync message and build the radio queue consisting of energy bursts;
3. When the interrupt occurs, the timestamp will be recalculated, translated into energy bursts using the given encoding and replaced in the X-Sync message.

3.4 Precise Reception Timing of CTC Messages

Figure 3.5 shows the deterministic and non-deterministic delays of packet-level clock synchronization systems. It illustrates a transmitting node (TX) sending a X-Sync transmission to a receiving node (RX). All delays present in conventional clock synchronization systems can be neglected or compensated for using state-of-the-art methods. X-Sync employs novel strategies to further compensate the non-deterministic delays inherent to packet-level CTCS systems, the averaging delay $\tau_{averaging}$ and the sampling delay $\tau_{sampling}$. The reasons for the former, as well as novel methods for its compensation are discussed in Section 3.4.1, which allow for estimation of $\tau_{averaging}$ without increasing the number of packets sent. The reasons for the latter are discussed in Section 3.4.2; For their compensation, an additional synchronization preamble is appended to the CTC transmission, which is also discussed in Section 3.4.2.

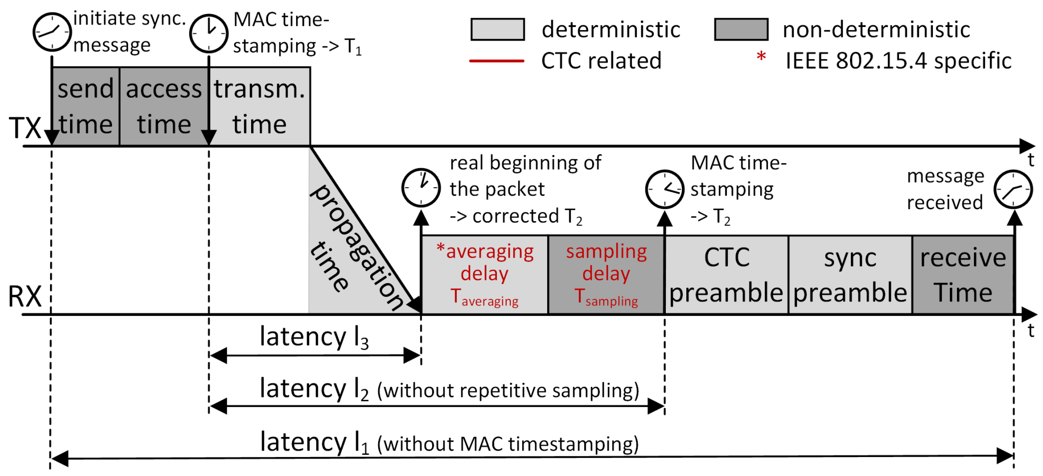


Figure 3.5: Transmission delay within X-Sync, divided into deterministic, non-deterministic, CTC related, and IEEE 802.15.4 specific delays. In this scenario, a transmitting node (TX) sends a synchronization message to a receiving node (RX). The CTC related sampling delay $\tau_{sampling}$ and the IEEE 802.15.4 specific averaging delay $\tau_{averaging}$ are further defined and described in Section 3.4.2 and 3.4.1 respectively.

3.4.1 Averaging delay $\tau_{averaging}$

In X-Burst [20], instantaneous and non-instantaneous RSSI measurements are defined and discussed. Their respective radios are defined in the same way as in this thesis. BLE and IEEE 802.15.4 make use of instantaneous and non-instantaneous RSSI sampling, respectively. Figure 3.6 shows an example of a X-Sync transmission on a non-instantaneous RSSI radio. It shows that a moving average filter is used to filter the RSSI values. In addition to that, if the RSSI querying speed is faster than the sampling speed, the same RSSI value is retrieved twice. This has several implications:

- The energy burst durations strongly depend on the chosen threshold.
- The beginning of the energy burst (rising edge) needed for the proposed CTCS also depends on the chosen threshold.

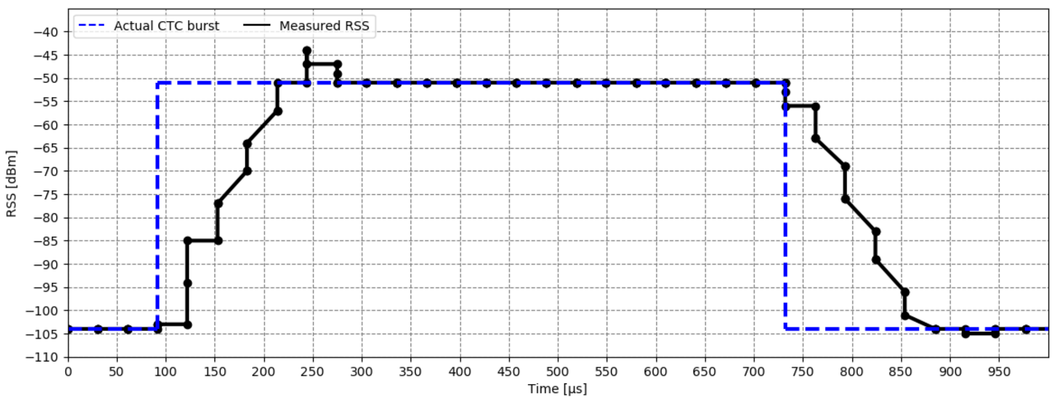


Figure 3.6: X-Burst transmission with an IEEE 802.15.4 receiver, whose RSS values are filtered by a moving average filter, in compliance with the standard. The difference between the received RSSI (solid, black), and the actual energy on the channel (dashed, blue), varies greatly.

The averaging delay $\tau_{averaging}$ is defined in Figure 3.7 as the timespan between the nominal start of the burst if no averaging would be applied and the actual detection of the energy burst. This delay has the following properties:

- Threshold dependent. The averaging delay $\tau_{averaging}$ is proportional to the configured RSS threshold.

- Location dependent. With decreasing distance between the transmitter and the receiver, the maximum RSS of the energy burst increases. This decreases the averaging delay $\tau_{averaging}$.
- Transmission power dependent. With decreasing transmission power, the averaging delay $\tau_{averaging}$ is also decreased.
- Non-deterministic.
- $\tau_{averaging}$ is in the interval $0 < \tau_{averaging} < 128\mu s$, because of the moving average filter defined in the IEEE 802.15.4 standard, which averages over exactly 8 samples.

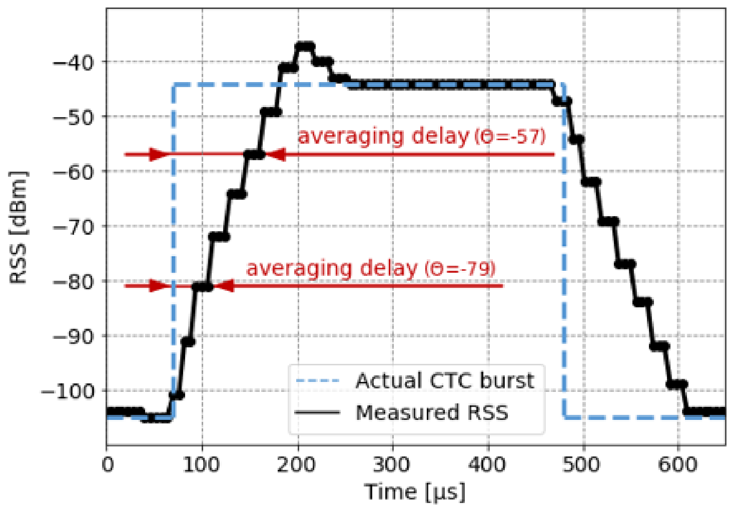


Figure 3.7: RSS measured at an IEEE 802.15.4 node (solid, black) versus the actual RSS values seen at the channel without any averaging (dashed, blue). The $\tau_{averaging}$ delay is defined as the interval between these two graphs and varies depending on the selected RSS threshold.

Without further compensation, the achievable synchronization accuracy would be limited by the change of the averaging delay between packets, either due to changing transmission powers, or due to changed distances between the transmitter and the receiver.

Averaging delay $\tau_{averaging}$ compensation

In order to define a compensation scheme for the averaging delay $\tau_{averaging}$, a distinction between `static` and `dynamic environments` has to be made. The former is defined as node arrangements where the position of the nodes is not changed during the synchronization. Additionally, no change in the transmission power is made. The latter covers the general case where every node may be moved and the transmission power can be changed freely.

Static Environments. For each transmitter-receiver pair (i,j), it exists exactly one averaging delay $\tau_{averaging,i,j}$ (which is therefore constant). This constant delay can be measured once when the physical arrangement of the nodes is fixed and can then be easily compensated for by subtracting this interval from the estimated clock at the receiver.

Dynamic Environments. The averaging delay between each transmitter-receiver pair is varying between synchronizations. This non-deterministic delay can vary by as much as $128\mu s$, which directly influences the achievable synchronization accuracy. When we assume short-term static conditions during the transmission of the preamble, and assuming that we know the difference between the measured duration $d_{measured}$ and the nominal duration $d_{nominal}$ of the energy burst, this delay can be estimated using the following formula:

$$\tau_{averaging,estimation} = 64\mu s - \frac{(d_{measured} - d_{nominal})}{2} \tag{3.1}$$

By continuously sampling the energy on the cross-technology channel and evaluating this energy with a predefined threshold, the measured duration $d_{measured}$ is known. Based on this duration, the nearest nominal duration $d_{nominal}$ can be calculated because only a distinct duration alphabet is allowed. With this information, the estimation can be calculated for every energy burst in the preamble. The average of those $\tau_{averaging,estimation}$ can then be subtracted from the estimated clock at the receiver.

3.4.2 Sampling delay $\tau_{sampling}$

All energy sensing CTC schemes are sampling the transmitted energy burst in software, either in the time domain or in the energy domain. The sampling and processing speed is therefore limited by the used hardware and is in the range of $20\mu s$ to $60\mu s$ per measurement, which is equivalent to a sampling rate of 50 to 16 kHz respectively. This has several implications for the packet-level CTC sampling:

- The sampled energy burst durations are not equal to the nominal durations. Instead, they can be off by as much as the RSSI sampling time.
- These variations are highly varying and unpredictable.
- Sampling the energy burst durations with a sampling frequency which is lower than twice the rise time of the energy burst effectively introduces artefacts because of Nyquist's sampling theorem.

Due to the low sampling rate of the RSS information, the detected start of the energy burst, when the set RSS threshold is exceeded, is not equal to the real start of a transmission. This can also be seen in Figure 3.8, which shows the definition of the sampling delay in more detail. We call this interval sampling delay $\tau_{sampling}$, which has the following properties:

- Unpredictable and non-deterministic.
- Depending on the exact sampling point in time, $\tau_{sampling}$ is highly varying between CTC transmissions.
- $\tau_{sampling}$ is in the interval $20 < \tau_{averaging} < 60\mu s$ for all used hardware platforms in this thesis.
- $\tau_{sampling}$ can never be negative, because detecting the start of the energy burst on the channel before it is sent is not possible.

Without compensation schemes, the achievable accuracy will always be limited by this delay and by the RSSI sampling speed of the node. Compensation methods are needed to achieve the required accuracy goal of $2.3\mu s$.

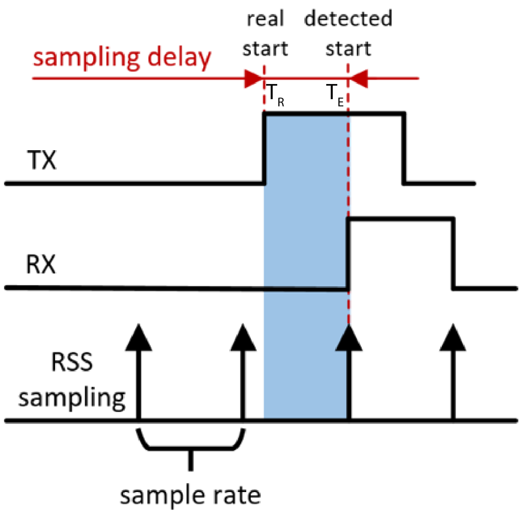


Figure 3.8: Transmission of one energy burst between one transmitter (TX) and one receiver (RX). The sampling delay $\tau_{sampling}$ is defined by the timespan between the real start of the burst on the physical channel and the detected start of the receiver $\tau_{sampling} = T_E - T_R$. This delay is caused by the low RSS sampling speed which is inherent in packet-level CTC solutions.

Sampling delay $\tau_{sampling}$ compensation

The RSSI sampling and processing frequency is defined by the underlying hardware and cannot be increased without changing the development platform. However, the specific point in time when the RSSI sampling is initiated can be chosen freely and with μs accuracy. The idea of X-Sync is to use this property to compensate for the low RSSI sampling frequency.

A X-Burst message consists of energy bursts of variable lengths, separated by variable breaks that can also be used for data encoding, if configured accordingly. Sending a known synchronization sequence, including known gaps between energy bursts, enables sampling with a sub-Nyquist frequency with the proposed algorithm. The basic idea is that, instead of sampling in real-time, the rising edge (the beginning of an energy burst) is repeated and can be sampled consecutively. This is also called repetitive sampling.

The receiving node hereby knows the nominal durations of both the energy bursts and gaps at compile-time and can calculate the beginning of each

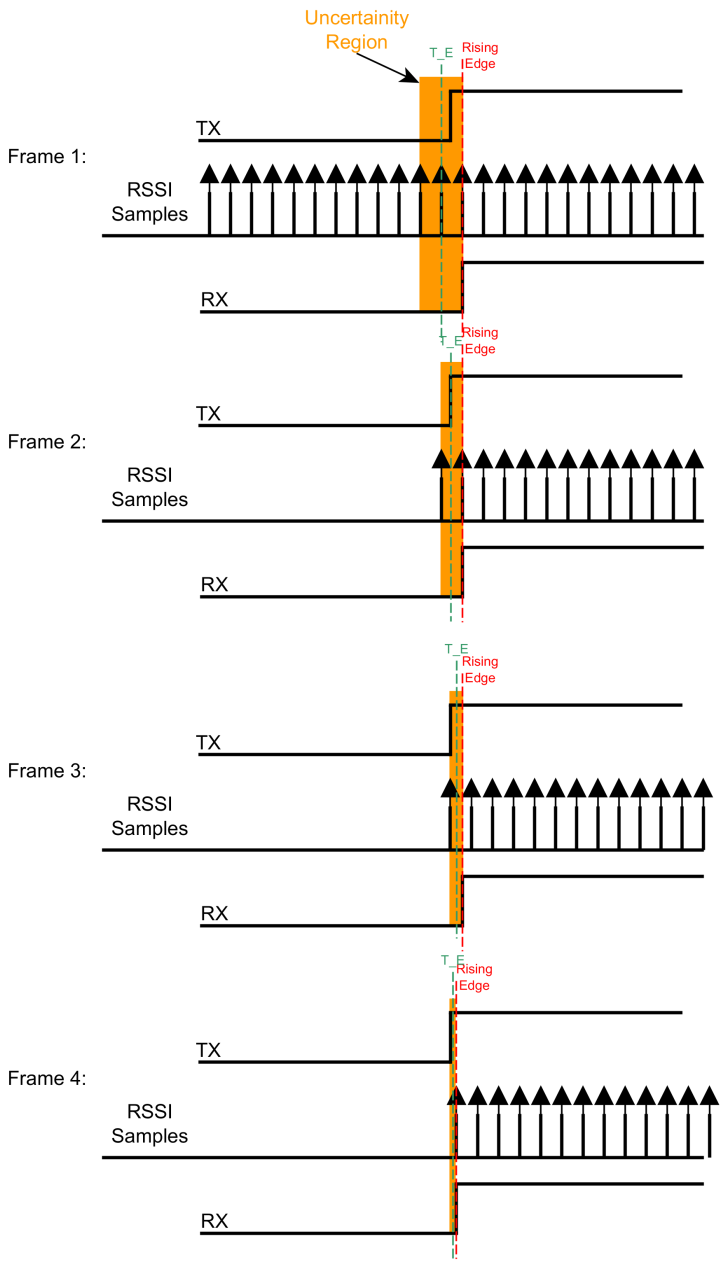


Figure 3.9: Timeframes of a X-Sync preamble used to compensate the varying sampling delay $\tau_{sampping}$ by applying a binary search algorithm.

energy burst in the synchronization preamble. We further refer to each of these events as frames. Efficient searching for rising edges within frames is done using the **binary search algorithm**. It works with a worst case complexity of $O(log2(n)+1)$, where n is the number of energy bursts in the synchronization preamble.

Figure 3.9 shows the implementation of the binary search algorithm within consecutive frames. Each frame shows the uncertainty region, which will get halved as the algorithm moves on. In the initial frame, the continuous signal will be sampled with a limited RSSI sampling speed to get a a crude estimation for the searched rising edge T_R by sampling with the given sampling speed. T_R will now be within

$$T_E^0 - \mathrm{c}^0 \leq T_R \leq T_E^0 \tag{3.2}$$

where T_E^0 is the initial estimated beginning of the energy burst and $\mathrm{c}^0 = \frac{1}{f_{sample}}$ is the correction based on the sampling frequency. The superscript index hereby represents the current frame index/iteration. By using this inequality, the estimation for the next timeframe T_E can be approximated to be exactly in the middle of T_E and $T_E - \mathrm{o}^0$, which results in a new estimation

$$T_E^1 = T_E^0 - \mathrm{c}^1 \tag{3.3}$$

The correction c needed for the next iteration is hereby $c^1 = (c^0/2$, where f_{sample} is the RSSI sampling speed of the X-Sync node.

In the next iteration, the signal is sampled exactly at this point in time. If the result indicates that the sampling time was before the rising edge (leading), the remaining interval shortens to

$$T_E^1 \leq T_R \leq T_E^0 \tag{3.4}$$

else (lagging) the remaining interval is

$$T_E^0 - \mathrm{c}^0 \leq T_R \leq T_E^1 \tag{3.5}$$

where the correction variable is halved again and for each following iteration:

$$\mathrm{c}^{i+1} = \frac{\mathrm{c}^i}{2} \tag{3.6}$$

and

$$T_E^{i+1} = \begin{pmatrix} T_E^i - c^i \; if \; leading \\ T_E^i + c^i \; if \; lagging \end{pmatrix} \tag{3.7}$$

These steps can be repeated until the uncertainty region is sufficiently small. The number of timeframes n needed for a given accuracy t_a is given by:

$$n = log_2(\frac{1}{f_{sample} * t_a}) \tag{3.8}$$

If we assume f_{sample} to be at the worst case within all used hardware platforms, at 16kHz, and t_a to be as low as $1\mu s$, the number of repetitive timeframes, and therefore energy bursts needed, results to $5.96 \approx 6$.

Figure 3.10 shows the $\tau_{sampling}$ compensation principle on a message level. As soon as the first energy burst after the preamble is detected, a binary search estimation is started in the consecutive known timeframes. At the end of the X-Sync preamble, the sampling delay will be compensated down to the chosen accuracy.

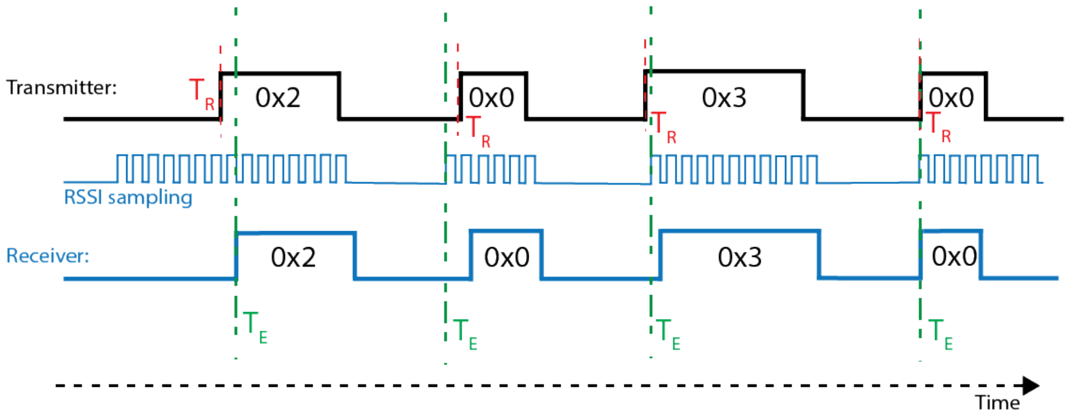


Figure 3.10: X-Sync sample delay $\tau_{sampling}$ compensation strategy shown without the division into separate timeframes. T_R indicates the start of the energy burst at the transmitter and T_E is the current estimate of T_R.

3.5 Validation and Synchronization

Due to the nature of packet-level CTC communication, transmissions are error-prone in terms of data integrity and timing integrity. X-Sync changes the data integrity checks used in X-Burst to allow cyclic redundancy checks. This filters the transmitted data to increase the data integrity. Assuming that the data integrity is sufficiently checked, the result of a timestamp transmission is forming a synchronization pair consisting of the timestamp at the start of the transmission on node A (e.g., T_1) and the timestamp at the reception on node B (e.g., T_2).

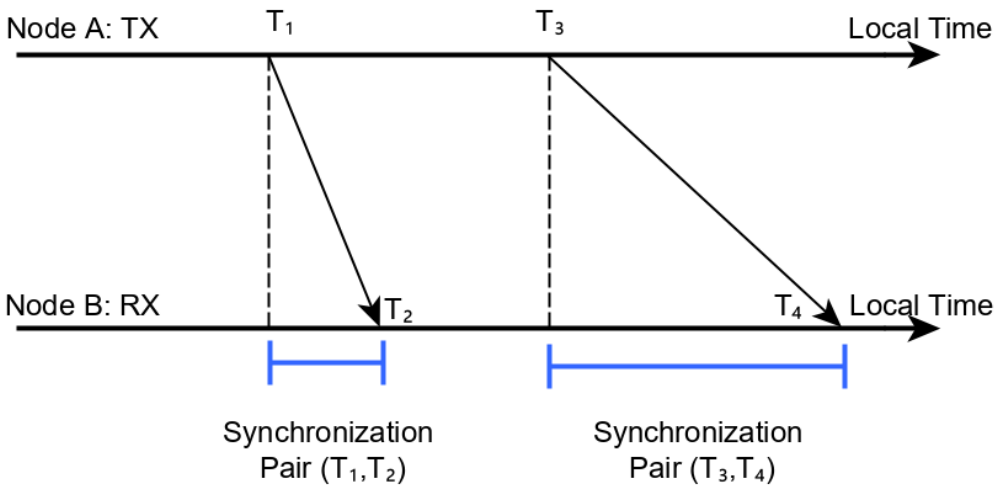

Node A: TX
T_1
T_3
Local Time
Node B: RX
T_2
T_4
Local Time
Synchronization Pair (T_1,T_2)
Synchronization Pair (T_3,T_4)

Figure 3.11: Consecutive timestamp transmissions between the transmitting node A and the receiving node B using X-Sync. Each successful transmission results in a synchronization pair consisting of the transmitting timestamp, the receiving timestamp, and the transmission delay.

These synchronization pairs could be used to correct the clock of Node B upon each arrival. Figure 3.12 shows the result of this synchronization approach assuming that a perfect synchronization is happening every second. Between these synchronization points, the error between two clocks will grow linearly over time, until the next synchronization is happening, because of the clock skew difference between Node A and B. Typically, these crystals have an accuracy α ranging from +40ppm to -40ppm [23] assuming constant temperature and ignoring second order effects, such as ageing. So, the following formula describes the drift d of two synchronized nodes as time progresses.

$$d(t) = t\ (\alpha_1 - \alpha_2) \tag{3.9}$$

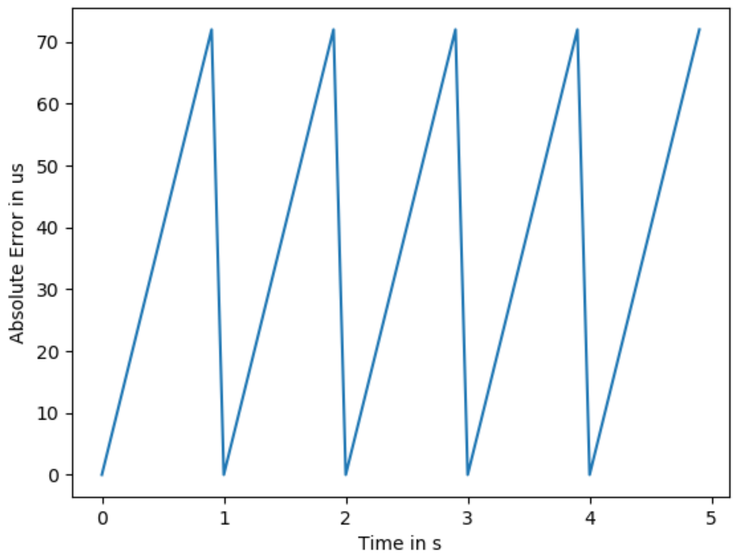


Figure 3.12: Simulated synchronization error assuming ideal synchronization every second and a typical clock skew of 40ppm without clock skew compensation.

It can be seen that clock skew estimation is therefore needed in every accurate clock synchronization system. X-Sync has additional requirements on the filtering of clock synchronization pairs because of the used packet-level transmissions and the proposed transmission delay compensation method. Figure 3.13 shows a sample transmission with a disturbance during time-frame 2 on the channel during the X-Sync synchronization preamble. This disturbance can either lead to a detection of an invalid X-Sync synchronization preamble, or to a slight variation within the start of the used energy bursts in the frames. In this case, the preamble will be detected to be valid, and the binary search algorithm will compensate T_2 with wrong values, leading to an invalid synchronization pair.

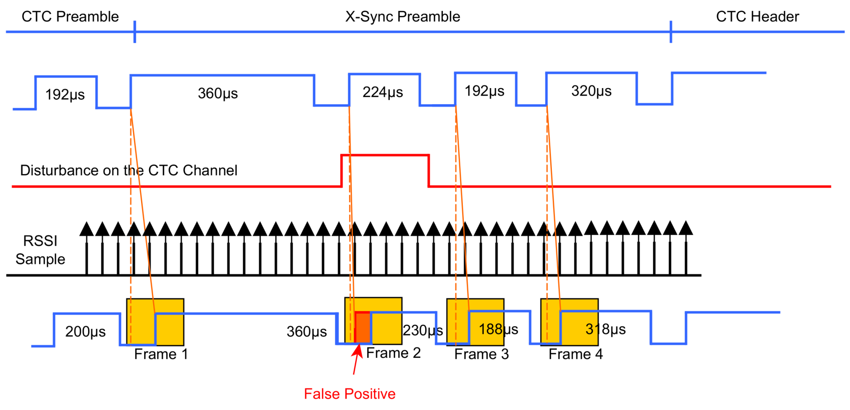


Figure 3.13: Disturbance on the CTC channel during an ongoing X-Sync transmission. T_{burst} is the nominal burst duration, which is set to $224\mu s$ in this example. The received energy burst is superposed by the channel disturbance, which results in a measured duration of $230\mu s$. It can be seen that the rising edge of the transmission T_R is obfuscated and the binary search algorithm will decide for the wrong side of the interval. This will result in an outlier, which has to be filtered by the upper clock synchronization layers.

Another important fact to consider is that, ideally, for any regression algorithm, all past synchronization pairs (T_1, T_2) should be stored. In reality, the available memory is constrained, which limits the amount of data that can be stored on a wireless sensor node. The common solution to this problem is to maintain a moving window buffer, where the last N elements are stored. If a new pair is received, the oldest stored datapoint is deleted and the new pair is added, in a first in-first out (FIFO) manner. Limiting the number of elements stored will have a high impact on the synchronization accuracy. Once these buffer is filled with synchronization pairs, a clock skew estimation algorithm can be used to estimate the linear clock skew difference.

3.5.1 Linear Regression

One way of fitting a line into N given synchronization points using linear regression $\{\{x_1, y_1\}....\{x_N, y_N\}\}$ is the least squares algorithm. The primary assumption is that the regression error is defined as:

$$E(x,y) = \sum_{n=0}^{N} (y_n - (ax_n + b))^2 \tag{3.10}$$

Using linear algebra, the minimum values for a and b can be found, leading to this equation:

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum_{n=0}^{N} x_n^2 & \sum_{n=0}^{N} x_n \\ \sum_{n=0}^{N} x_n & \sum_{n=0}^{N} 1 \end{pmatrix} \begin{pmatrix} \sum_{n=0}^{N} x_n y_n \\ \sum_{n=0}^{N} y_n \end{pmatrix} \tag{3.11}$$

Albeit being the de-facto standard for linear estimation methods, this algorithm is prone to errors, due to the fact that it is based on the assumption that all data points are error-free. This behaviour can be seen in Figure 3.14, which shows how a near perfect dataset can lead to a wrong clock skew estimation if there is only one distinct outlier at t=45. Due to the eventual generation of invalid X-Sync synchronization pairs, this clearly shows that filtering is needed to ensure accurate and robust synchronization.

3.5.2 Thresholding

The simplest approach to filter the synchronization data is to set a fixed threshold at compile-time. By carefully selecting this limit, invalid datapoints can be filtered and only valid elements will be added to the moving window buffer, as shown in Figure 3.15. However, this threshold can vary and, if it is set too high, no packet will be stored in the moving window buffer. If it is set too low, however, every synchronization packet will be stored, including outliers.

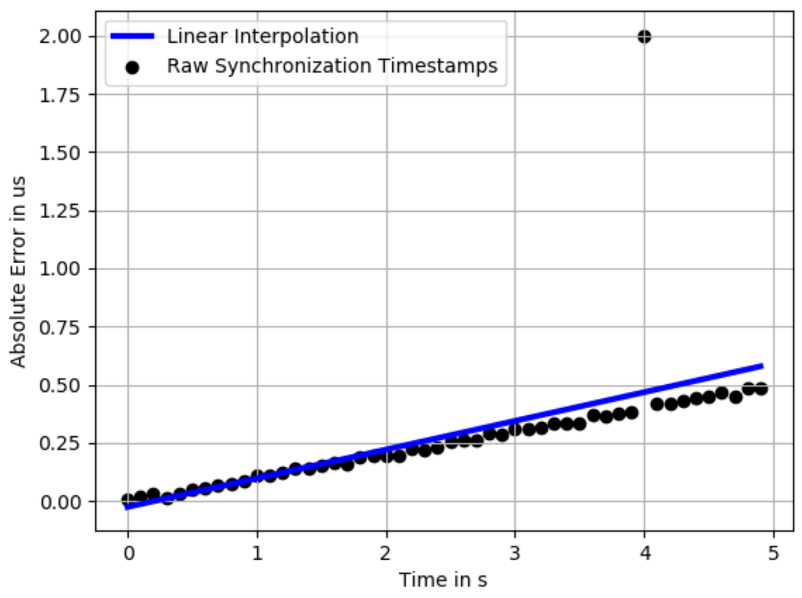


Figure 3.14: Simulation of linear regression with one outlier at t=45. It shows that one outlier in the dataset will greatly influence the clock skew estimation, and therefore, the achievable accuracy.

3.5.3 RANSAC

Random sample consensus (RANSAC) is an iterative method for differentiating inliers and outliers. It defines inliers as data whose distribution can be explained by the used model versus outliers as data which does not fit in the model [61]. The following algorithm is used to find valid samples (inliers) in the one dimensional case with resource constraints in mind:

1. Select a subset of the moving window buffer, called hypothetical inliers. This set consists of two datapoints for the one dimensional case.
2. Fit a simple model to the found set. Due to the fact that embedded hardware imposes resource constraints, the simplest one dimensional

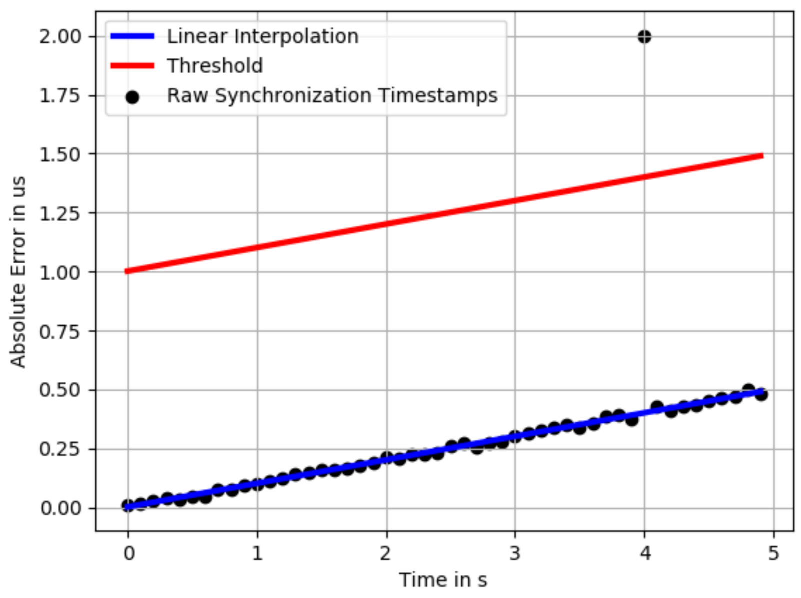


Figure 3.15: Simulation of linear regression with one outlier using a threshold filter. In this example, the outlier will be filtered, and the result is again nearly ideal.

model was used:

$$a = \frac{y_2 - y_1}{x_2 - x_1}, \ b = a\,(x_2 - x_1) + y_1 \tag{3.12}$$

where (x_1, y_1) is the first chosen synchronization tuple, and (x_2, y_2) the second.

3. All other data is tested against this model described by a and b using a model specific loss function. In case of X-Sync this is simply a thresholding function. The number of valid samples is stored for comparison.
4. If the maximum number of repetitions is reached, the model with the most number of samples is used. Otherwise, the algorithm is continued at 1.

The results of this modified version of RANSAC are the linear model parameters of the best fit as well as a differentiation between outliers and inliers. In addition to that, the number of outliers gives a rough estimation for the sample quality.

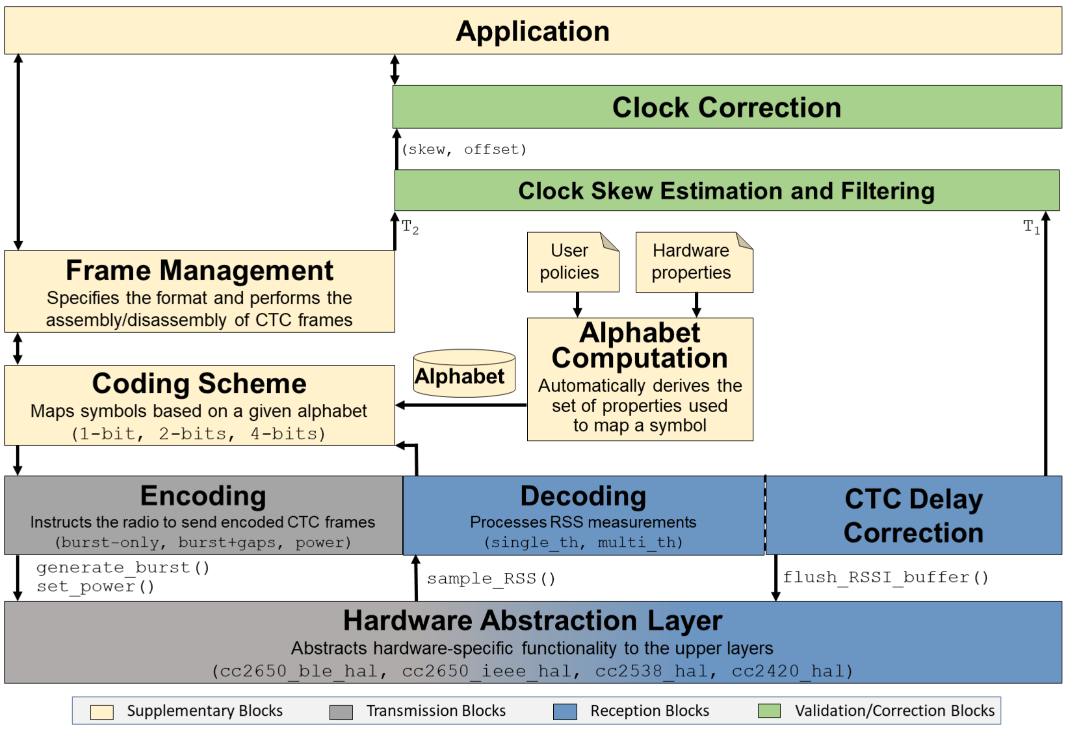
In general, RANSAC is used for robust, error-prone data filtering. However, the best results are achieved when the maximum number of repetitions is very high, which is not ideal for constrained embedded nodes.

3.5.4 RANSAC and Linear Regression

The downside of using this variant of RANSAC is that the resulting model is only defined using two points, although all inliers could theoretically be used to estimate the parameters a and b. In this approach, a least squares fit algorithm was applied to all inliers to further improve the result.

4 X-Sync: Implementation

The following sections explain the implementation-specific details of the X-Sync cross-technology clock synchronization scheme. To reduce the complexity of the proposed solution, we implement X-Sync as an extension of X-Burst [42] using a highly-modular architecture, which maximizes code reuse and increases the portability to other hardware platforms. Figure 4.1 shows the proposed architecture that divides the application into different blocks grouped by supplementary, transmission, reception and validation/-correction blocks.



Application
Clock Correction
(skew, offset)
Clock Skew Estimation and Filtering
T2
T1
Frame Management
Specifies the format and performs the assembly/disassembly of CTC frames
User policies
Hardware properties
Alphabet
Alphabet Computation
Automatically derives the set of properties used to map a symbol
Coding Scheme
Maps symbols based on a given alphabet (1-bit, 2-bits, 4-bits)
Encoding
Instructs the radio to send encoded CTC frames (burst-only, burst+gaps, power)
Decoding
Processes RSS measurements (single_th, multi_th)
CTC Delay Correction
generate_burst()
set_power()
sample_RSS()
flush_RSSI_buffer()
Hardware Abstraction Layer
Abstracts hardware-specific functionality to the upper layers (cc2650_ble_hal, cc2650_ieee_hal, cc2538_hal, cc2420_hal)
Supplementary Blocks
Transmission Blocks
Reception Blocks
Validation/Correction Blocks

Figure 4.1: The highly modular implementation of X-Sync extending X-Burst [42], divided into transmission, reception, validation/correction and supplementary blocks.

Supplementary blocks (Section 4.1) are needed to pre/post-process X-Sync transmissions. These blocks include the application, the frame management, the coding scheme, as well as the chosen communication alphabet and include only minor changes from the original X-Burst blocks. The transmission blocks (Section 4.2) are divided into an encoding block and the HAL. These blocks help to achieve accurate transmission timings, needed for synchronization. For receiving X-Sync transmissions and achieving accurate reception timings, the reception blocks (Section 4.3) are divided into a decoding block, a CTC delay compensation block and the HAL. The received time synchronization pairs are further filtered and processed in the validation/correction blocks (Section 4.4), consisting of a clock skew estimation and filtering block, as well as a clock correction block.

4.1 Supplementary Blocks

In this section, the supplementary blocks are described and discussed. All blocks in this section are needed to construct/deconstruct X-Sync messages, and their operation is not time-critical and fully hardware agnostic. Hereby, all blocks except the frame management block are directly taken from X-Burst [42] and will be described for reference reasons only. The frame management block was changed to establish a new message format, which includes the synchronization preamble, changes in the CTC header, and the actual timestamp sent from the transmitter T_1.

4.1.1 Application Block

This block defines the application logic, which uses X-Sync for cross-technology clock synchronization. Sensors and actuators can be made available to the distributed network, and nodes can furthermore handle complex tasks required by the application. Regarding synchronization, the application can set the synchronization interval, the exact moment in time when the synchronization happens, and the number of synchronization packets that a X-Sync message consists of. All these parameters are depending on the synchronization accuracy, memory consumption, and energy consumption requirements.

X-Sync packets can be constructed and sent at any time by specifying header and payload (if any). The header, described in Section 4.1.2, includes a flag that adds the synchronization information to the CTC packet. The lower blocks abstract and hide the complex mechanisms during transmission and reception from the application block. Upon reception of a X-Sync transmission, the application block gets notified using a Contiki-NG event and can access the synchronized clock. To summarize, the application block interacts with the lower blocks using the following two actions:

- **Specify header, payload and trigger a X-Sync transmission.** The CTCS functionality is embedded into Contiki-NG and follows the specified calling conventions. The following shows the construction of a header, where a timestamp and a checksum should be appended to the transmitted data.

  ```
  struct ctc_header header = {0};
  header.bIncludeTimestamp = true;
  header.bIncludeChecksum = true;
  NETSTACK_RADIO.set_object(RADIO_PARAM_CTC_HEADER, &header,
   ↪  sizeof(header));
  ```

  Once the application block decides to trigger a new X-Sync synchronization, the following call will queue a new transmission, which then gets processed by the frame management block. The transmission will not start immediately, depending on the chosen radio duty cycle, as described in detail in [20].

  ```
  NETSTACK_RADIO.set_object(RADIO_PARAM_CTC_TX_DATA, ctc_data,
   ↪  sizeof(ctc_data));
  ```

- **Receive CTC messages and access the synchronized clock.** In compliance with the Contiki-NG event system, the application detects a ctc_event including a parameter containing payload and the timing information. The following shows a typical use-case:

```
while(1)
      {
              PROCESS_WAIT_EVENT_UNTIL(ev == CTC_EVENT);
              struct ctc_rx_data rx_data = *(struct
               ↪  ctc_rx_data*)data;
      }
```

4.1.2 Frame Management Block

The frame management block assembles and disassembles X-Sync messages using the structure shown in Figure 4.2.

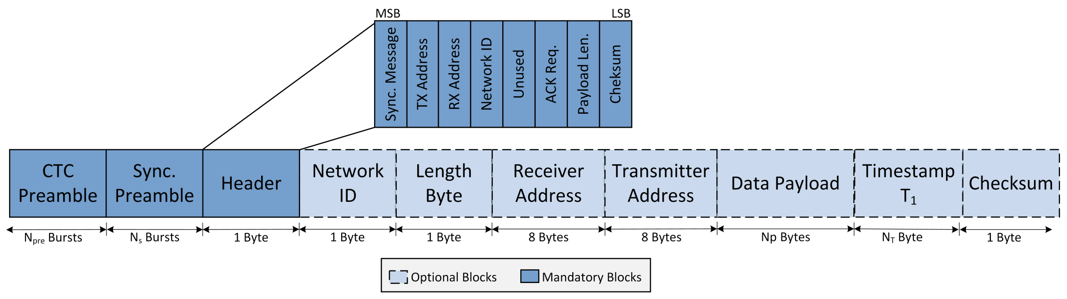


Figure 4.2: High-level structure of a X-Sync message divided into mandatory and optional parts. Every message starts with the CTC preamble, which allows receiving nodes to detect a beginning of a transmission. The frame management block adds a synchronization preamble to allow compensation of the delays inherent in X-Sync. Next, the header defines the optional fields included in the message, such as the network ID, the length byte, the receiver and transmitter addresses, a timestamp, and the checksum. If a field is enabled, it will get appended to the message accordingly.

At the beginning of a X-Sync message, the CTC preamble is sent, which allows receiving nodes to detect the CTC message on the shared cross-technology channel. To make that possible, receiver and transmitter both know the amount of (N_{pre}) and the duration of each CTC preamble burst at compile-time. Next, the message includes a synchronization preamble that allows compensation of the delays inherent in a packet-based CTCS system, as described in Section 3.4. The number of synchronization bursts N_s can be chosen depending on the synchronization accuracy requirements. Section 5.2 will evaluate the impact of N_s on the synchronization accuracy. Before the actual data and the timestamp T_1 can be transmitted, a header gets added to specify which fields are included in the X-Sync messages. After the CTC header, the payload is added (if any). The fields of the header and their purposes are described in more detail below:

- **Synchronization Message:** Appends the timestamp T_1 to the X-Sync message. Without this information, no clock synchronization is possible at the receiver. Thereby, the size of timestamp T_1 is configurable, depending on the chosen clock source of the transmitter.

- **TX Address:** Appends a unique 8-byte transmission address to the X-Sync message. With this field, any receiving node can determine the origin of the message.
- **RX Address:** Enabling this flag and specifying a receiver address allows for unicast and broadcast transmissions. If the address of the receiving node does not match the RX address field, the message gets ignored by the frame management block. Furthermore, if the RX address field gets omitted, a broadcast can be created.
- **Network ID:** Setting this flag in the header will add a one-byte network ID field used to create virtual networks within the heterogeneous wireless sensor network. The frame management block will compare this ID to the assigned network ID of the receiver and will discard the message in case they do not match.
- **ACK Request:** When this flag is received, any message that is transmitted needs to be acknowledged by the receiver. The frame management block will thereby manage the reply.
- **Payload Length:** By specifying this flag, any receiver can check if the received message length equals the sent message length used primarily for validation purposes.
- **Checksum:** By setting the checksum flag, a CRC8 checksum byte is appended at the end of the transmission to validate the correct reception of the message.

4.1.3 Alphabet Communication Block

Both receiver and transmitter must use the same communication alphabet in order to be able to communicate with each other over the cross-technology channel. The alphabet block is directly taken from X-Burst [20], and it is precomputed for the user policies and the hardware properties. User policies allow the user to set the focus on reliability or throughput, while the node's hardware properties are purely defined by its hardware capabilities. IEEE 802.15.4 and BLE radios cannot transmit packets with arbitrary transmission durations but only distinct durations. The authors of [20] show that the following equations give the durations of IEEE 802.15.4 transmissions:

$$d_Z(n) = \sigma_Z * n = \frac{1}{250000} * n = 4 * 10^{-6} * ns; \tag{4.1}$$

$$d_{Z,min} = 192\mu s \tag{4.2}$$

$$d_{Z,max} = 4256\mu s \tag{4.3}$$

where n is the number of sent bytes. Likewise, for BLE, the following equations hold:

$$d_Z(n) = \sigma_{BLE} * n = \frac{1}{1000000} * n = 1 * 10^{-6} * n \tag{4.4}$$

$$d_{BLE,min} = 80\mu s \tag{4.5}$$

$$d_{BLE,max} = 2120\mu s \tag{4.6}$$

With the use of BLE version 4.2 or lower, the maximum BLE transmission duration $d_{BLE,max}$ can only be achieved by the use of BLE test packets, whose length is limited to 255 bytes (2120μs on the TI CC2650). Later versions of BLE allow for longer transmissions using normal messages. With these limitations, a common set of durations (and therefore packet lengths) between IEEE 802.15.4 and BLE radios, optimized for throughput, is given as an example in Table 4.1. Table 4.2 shows a similar communication alphabet focused on reliability, also between IEEE 802.15.4 and ZigBee devices.

4.1.4 Coding Scheme Block

The coding scheme block maps the bytes of the X-Sync message into symbols which can be 1-bit (2 durations), 2-bit (4 durations) and 4-bit (16 durations) wide, i.e., one burst contains 1,2 or 4 bits of information. This block is taken from X-Burst [42]. These schemes work by dividing each byte of the X-Sync message into groups of 1, 2, or 4 bits, as schematically shown in Figure 4.3.

Energy Burst Duration [μs]	Payload Bytes BLE	Payload Bytes IEEE 802.15.4
192	14	0
224	18	1
256	22	2
288	26	3
320	30	4
352	34	5
384	38	6
416	42	7
448	46	8
480	50	9
512	54	10
544	58	11
576	62	12
608	66	13
640	70	14
672	74	15

Table 4.1: X-Sync communication alphabet with focus on throughput between IEEE 802.15.4 and BLE nodes, as defined in X-Burst [20].

Each group can then be translated into durations by the use of the alphabet defined in this block. Table 4.3 hereby shows a coding scheme for 1-bit, 2-bit and 4-bit symbols based on the encoding alphabet defined in Table 4.1.

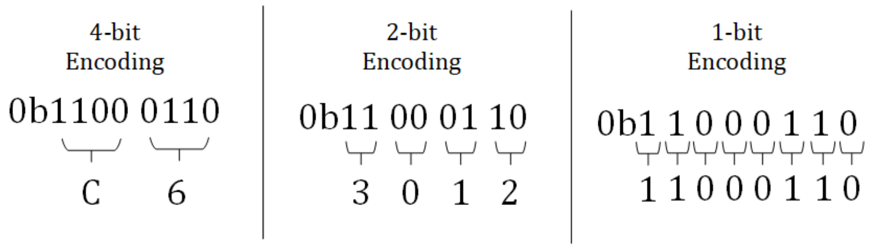


Figure 4.3: Division of binary data (0xC6) into symbols (shown in hexadecimal digits) using different encodings (4-bit, 2-bit, 1-bit).

To demonstrate the impact of different coding schemes, a simple one-byte payload (0xC6) is given. Figure 4.4, 4.5, and 4.6 show the translated energy bursts for a 1-bit, 2-bit, and 4-bit encoding respectively. In this example, only the energy bursts are used to encode information, but information could

Energy Burst Duration [μs]	Payload Bytes BLE	Payload Bytes IEEE 802.15.4
192	14	0
288	26	3
384	38	6
480	50	9
576	62	12
672	74	15
768	86	18
864	98	21
960	110	24
1056	122	27
1152	134	30
1248	146	33
1344	158	36
1440	170	39
1536	182	42
1632	194	45

Table 4.2: X-Sync communication alphabet with focus on reliability between IEEE 802.15.4 and BLE nodes, as defined in X-Burst [20].

also be encoded within the gap between consecutive bursts, as described in [42].

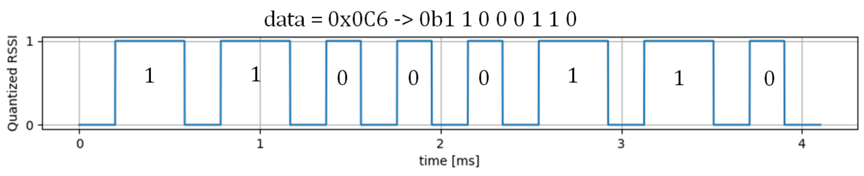


Figure 4.4: The mapping to energy bursts using the 1-bit coding scheme. Table 4.3 shows the used alphabet, and the information is encoded using energy-bursts only.

Energy Burst Duration [μs]	4-bit Coding Scheme	2-bit Coding Scheme	1-bit Coding Scheme
192	0x0	0x0	0x0
224	0x1	0x1	0x1
256	0x2	0x2	
288	0x3	0x3	
320	0x4	0x4	
352	0x5		
384	0x6		
416	0x7		
448	0x8		
480	0x9		
512	0xA		
544	0xB		
576	0xC		
608	0xD		
640	0xE		
672	0xF		

Table 4.3: Coding scheme for 1-bit, 2-bit and 4-bit symbols based on the encoding alphabet defined in Table 4.1.

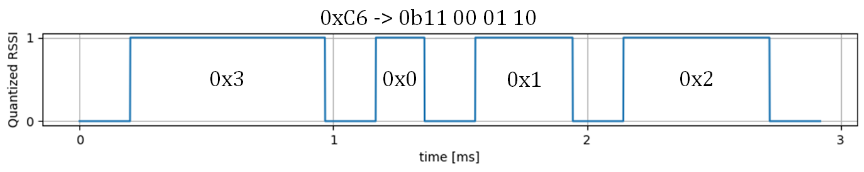


Figure 4.5: The mapping to energy bursts using the 2-bit coding scheme. Table 4.3 shows the used alphabet, and the information is encoded using energy-bursts only.

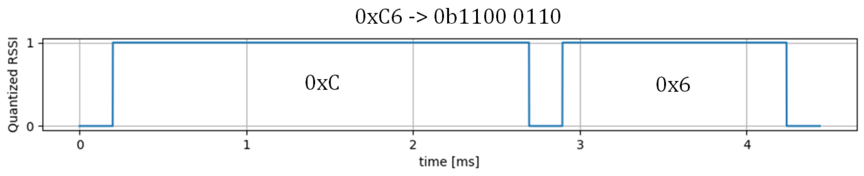


Figure 4.6: The mapping to energy bursts using the 4-bit coding scheme. Table 4.3 shows the used alphabet, and the information is encoded using energy-bursts only.

4.2 Transmission Blocks

This section groups all blocks relevant for the time-accurate transmission of X-Sync messages. This includes the encoding block, which translates the X-Sync messages into durations and also manages the transmission power. Followed by that, the transmission part of the hardware abstraction layer is described in detail.

4.2.1 Encoding Block

The encoding block receives all X-Sync durations as a parameter, processes them, and passes all bursts including information about the gap between two consecutive energy bursts, one-by-one to the HAL. The HAL only needs to know how long the current energy burst is, as well as the duration of the next gap. Depending on which part of the X-Sync message is transmitted, the encoding block will choose a different encoding scheme:

- **CTC preamble:** The start of a CTC transmission on the shared channel is indicated by sending the CTC preamble. Receivers have to detect this preamble reliably also in the presence of noise on the channel. Therefore, the encoding block encodes the CTC preamble only into bursts, which lowers the susceptibility to noise, with the gaps in-between set to zero. When the HAL processes these bursts and gaps, it tries to keep the gaps as short as possible, with respect to the capabilities of the used radio. The actual gap duration between bursts can be varying and is not relevant for preamble detection at the receiver. Figure 4.7 shows an example CTC preamble with exact energy burst durations and varying gap durations in-between.
- **Synchronization preamble:** The transmission of the synchronization preamble is time-critical, and all durations (burst and gaps) need to be well defined and transmitted accurately. Therefore, the encoding block maps these durations to energy bursts with fixed, well-defined gaps in between. The worst-case radio processing time of the energy burst defines the duration of these gaps, which ensures that the radio can always prepare the next burst during a gap. The synchronization state

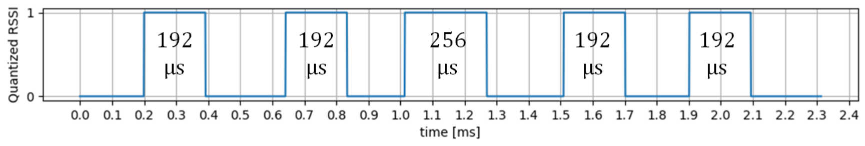


Figure 4.7: Example of a CTC preamble consisting of five energy bursts and varying gap durations in-between. Hereby, the following preamble durations were chosen: 192 μs, 192 μs, 256 μs, 192 μs, and 192 μs. The gap durations are not relevant for the correct detection of the preamble at the receiving node.

machine of the receiver must be ready to accept the next energy burst as well, which also limits the lowest possible gap delay. Figure 4.8 shows an example synchronization preamble with precisely defined gaps and bursts.

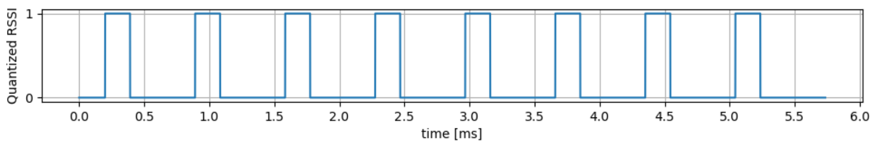


Figure 4.8: An example synchronization preamble consisting of eight energy bursts and precisely defined gaps in-between. Hereby, the shortest possible energy burst of 192 μs is repeated eight times with a well defined gap duration of 200 μs in between.

- **Optional Fields, Payload and Timestamp T_1:** The rest of the transmission is not time-critical and thus, the encoding block can use burst-only encoding or burst-and-gap encodings as defined in the X-Burst paper [42]. An example of a burst-only encoding is given in Figure 4.5 using the payload 0xC6 and the 2-bit encoding scheme. Figure 4.9 shows the same payload using the burst-and-gap encoding.

The resulting tuples (d_{burst}, d_{gap}) are then passed to the lower blocks one-by-one. In order to support HALs with radio queue support, the encoding block further signals a finished transmission, which will trigger the actual transmission at the HAL block. Moreover, the encoding block sets the output power depending on the configuration.

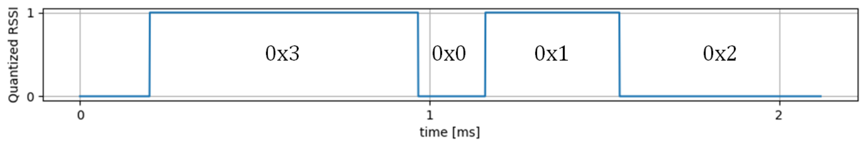


Figure 4.9: The payload (0xC6) mapped to energy bursts and gap durations in-between using the 2-bit coding scheme. Table 4.3 shows the used alphabet.

4.2.2 HAL - Transmission Part

The transmission section of the HAL block is the interface between the upper blocks and the actual radio hardware. Thereby, the responsibilities of the HAL are 4-fold:

- **Control radio transmissions using radio queues where available.** The underlying radio hardware could provide access to a radio queue, as mentioned in Section 3.3.2. Thereby, such radio queues do not implement a common standard, and access is platform-specific. Upon a call of the `send_burst` method from the encoding block with the tuple (d_{burst}, d_{gap}) as parameter, this burst is enqueued. As soon as the encoding block signals a finished transmission, the radio sends the whole queue using hardware-dependent mechanisms. Thereby, the radio completely takes control of the timing regardless of what is happening on the CPU.
- **Control radio transmissions by sending packets burst by burst.** If no radio queue is available on the hardware platform, each energy burst is sent individually at the cost of accuracy. With a call of `send_burst`, the parameter tuple consisting of (d_{burst}, d_{gap}) is immediately sent using native methods provided by the radio. This is implemented as a blocking call, so the CPU waits for a finished transmission. For sending the next tuple, the CPU will busy-wait for the amount of time specified for the gap d_{gap} and will then immediately send the next energy burst d_{burst}. The CPU must not be interrupted during the busy-wait phase, as this can negatively influence the transmission timing.
- **Manage ISR priorities to enable accurate transmission timing.** Most

CPUs implement support for ISR priorities. If this is the case, the HAL elevates the interrupt context of the transmission so that no other event can interrupt the ongoing transmission. Other platforms, such as the TelosB do not feature interrupt priorities and interrupts will simply be disabled during the busy-wait phase.

- **Implement MAC timestamping for cross-technology transmissions.** MAC timestamping uses different interrupts for timing that are different among hardware platforms. Such interrupts include transmission-started ISRs, x-bytes-transmitted ISRs and preamble-sent ISRs. Upon the invocation of such an event, the current time at the transmitter T_1 is sampled and appended to the ongoing transmission using the `append_mac_timestamp(ctc_clock_time_t` T_1`)` method.

Each node has to implement the following transmit method defined by the HAL:

- `send_burst(uint16_t burst_duration, uint16_t gap_duration, boolean last_packet)`

Additionally, each node has to define and call the following method for MAC timestamping in an ISR context, with a timestamp as parameter, sampled immediately when the ISR is invoked:

- `append_mac_timestamp(ctc_clock_time_t` T_1`)`

4.3 Reception Blocks

In this section, the block relevant for receiving X-Sync messages are described in detail. Thereby, the decoding block includes the CTC delay correction algorithms and builds upon the transmission part of the HAL layer.

4.3.1 Decoding and CTC Delay Correction Block

The output of this block is the corrected clock T_2 which gets passed to the upper blocks. Three steps are executed consecutively in order to correct T_2, which are described in detail below.

CTC Preamble Detection

Figure 4.10 shows the simplified state machine used for detecting the presence of a preamble on the CTC channel. First, the state machine calls the `sample_RSS()` method, provided by the HAL to get the current RSS value of the shared channel, which then gets compared against a certain static threshold known at compile-time. Once the value exceeds this threshold, the current clock is sampled. If the RSS is falling below the threshold again, the current clock gets sampled again, and the receiver calculates the duration of the energy burst. This duration gets added to the queue, and the whole buffer gets compared to the known preamble considering an allowed tolerance. if the measured durations match with the specified ones, the preamble is detected successfully and the next part of the X-Sync message can be received.

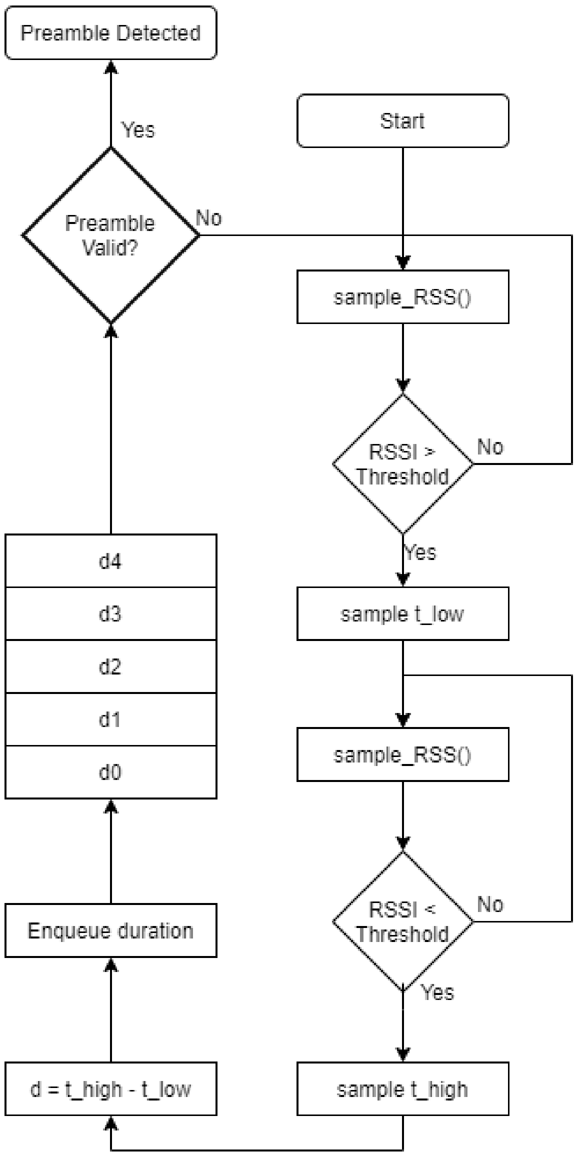


Figure 4.10: State machine for detecting the CTC preamble on the shared channel. Timeout checks are thereby not shown for simplification. First, the receiver samples the current RSS until a whole burst is detected, whose duration d gets calculated. This duration gets added to the queue, and the whole buffer gets compared with the known preamble. If the stored buffer and the known preamble are equal (including tolerances), the preamble is successfully detected, and the next part of the X-Sync message can be received.

Sampling Delay Compensation

The state machine given in Figure 4.11 implements the synchronization phase, which is triggered once the CTC preamble was successfully detected. The working principle of this complex state machine is described in Section 3.4.1 in detail and is divided into three phases:

- **Phase One: Initialization.** The first step in this phase is to initialize all variables. First, due to the fact that the hardware-dependent sampling frequency is known, the maximum correction that can be expected is $c^0 = \frac{1}{f_{sample}}$, as described in Section 3.4.1. Second, the durations of the energy bursts and the gaps of the synchronization preamble are known at compile-time and the packet durations p^i are defined as the sum of those (for each iteration i of the state machine).
  Once all variables are initialized, the state machine begins with searching for the beginning of the first energy burst of the synchronization preamble by using continuous RSS sampling. Once the receiver detects this point in time, it immediately samples the clock using the `CTC_TIMER_NOW()` method provided from the HAL which will be used as a starting point for the binary search algorithm. The receiver can then calculate the expected start of the next synchronization burst by adding the packet duration p^i.
- **Phase Two: Sampling.** In this phase, the actual sampling is taking place. The IEEE 802.15.4 radio averages the current RSS level internally, and therefore, this internal buffer has to be reset as a pre-requisite for precise detection of the next rising edge, i.e., the beginning of the next energy burst. Therefore, the HAL provides a method called `flush_RSS_buffer()` for IEEE 802.15.4 radios which handle this task. Next, precisely at the previously calculated timestamp T_E^i, the current RSS value on the channel will be sampled by calling `sample_RSS()`.
- **Phase Three: Calculation.** The sampled RSS value gets compared against the defined static threshold, and the result is either that the energy burst was not yet sent (leading) or is already present on the channel (lagging). The binary search algorithm then halves the uncertainty region as described in Section 3.4.1, and will be repeated until the last energy burst of the synchronization preamble was detected.

The procedure ends when the correction value is zero, which means that the binary search algorithm finished. After this process, timestamp T_2 is corrected and can be passed to the upper blocks for further processing.

Payload Reception

After the synchronization preamble, the receiver retrieves the payload and the timestamp T_1. The radio thereby samples either with a single threshold or with multiple thresholds as described in [42]. The former is used for non-averaging radios, like BLE. Figure 4.12 shows an example of an energy burst reception using a BLE radio. Hereby, defining a single threshold is sufficient to correctly determine the correct duration of the transmitted energy burst including small deviations. In the upper block, durations can then be easily mapped to symbols using burst-only or burst-and-gap encodings, a coding scheme and a predefined communication alphabet.

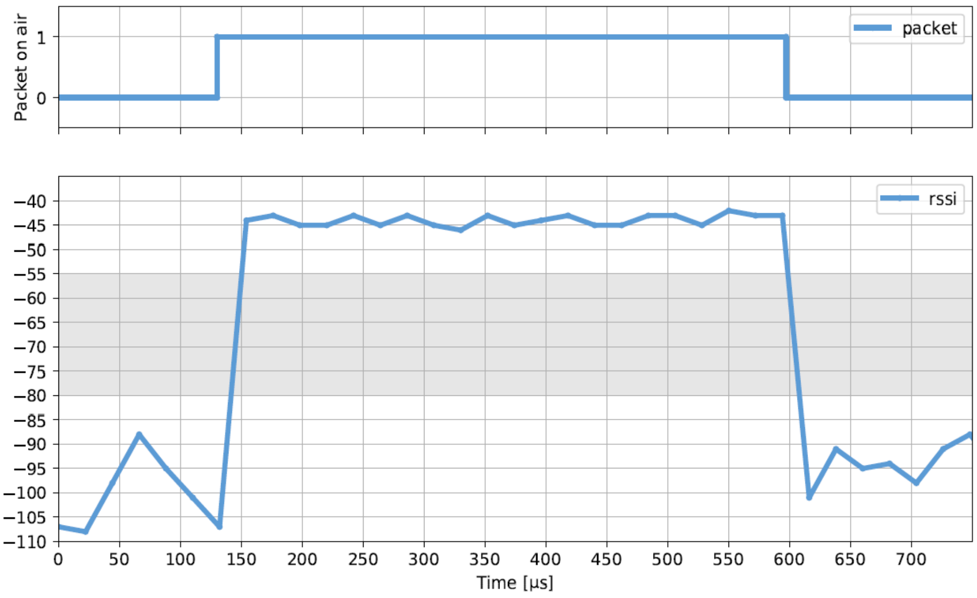


Figure 4.12: Energy burst reception on a BLE node, which is a non-averaging radio. Image taken from [20].

The latter is used for averaging radios, such as IEEE 802.15.4 ones. Figure 4.13 shows an example of an energy burst reception using an IEEE 802.15.4 radio. Hereby, one threshold is not sufficient to determine the correct duration of the energy burst with a certain tolerance. Choosing a high threshold will lead to significantly shorter durations (duration1), and likewise, choosing a low threshold will lead to longer durations (duration2). The authors of [20] therefore proposed to use multiple thresholds to increase the accuracy in determining the correct duration. The actual duration of the energy burst can then be calculated by averaging all given durations. These averaged durations, which are now well within tolerances, can then be translated to symbols again.

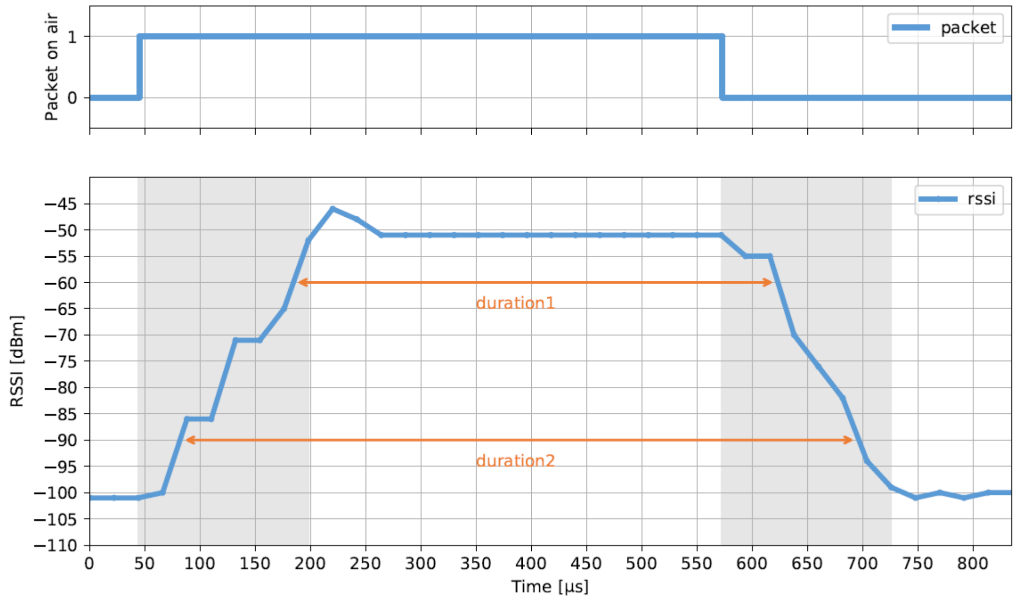


Figure 4.13: Energy burst reception on a IEEE 802.15.4 node, which is an averaging radio. Image taken from [20].

4.3.2 HAL - Reception Part

The receive section of the hardware abstraction layer consists of three tasks which the node has to fulfill:

- **Sample RSS values.** Upper blocks can call `sample_RSS()` in order to retrieve the current RSS value on the channel. Hereby, averaging radios, such as IEEE 802.15.4 ones, cannot provide the raw channel value and will instead return an averaged RSS information.
- **Flush the internal RSS buffer of the radio.** Averaging radios need to define a function which resets the radio's hardware RSS buffer. The IEEE 802.15.4 hereby does not provide a convenient way to achieve this behaviour. However, the radio's receive mode can be turned off or on at any time. Thus, turning the radio off and then turning it on again resets the buffer on all tested hardware platforms. Doing so takes some time, which is hardware dependent. Therefore, the gaps in the synchronization preamble have to be longer than this timespan in order to allow the radio to reset the internal averaging buffer. Higher blocks can precisely time this call of the `flush_RSS_buffer()` method in order to receive one non-averaged RSS value.
- **Provide a stable clock with a free-running timer.** The HAL needs to provide a stable and accurate clock as all upper blocks rely on having an accurate timer value that can be read at any time. Therefore, the upper blocks simply have to call `CTC_TIMER_NOW()` which will return a free-running timer value whose data-type is also configurable by the HAL (`ctc_timer_clock_t`).

Summarizing, each node has to implement the following methods in the receive section of the HAL:

- `sample_RSS()`. Samples and returns the current RSS value on the CTC channel.
- `flush_RSS_buffer()`. Flushes the internal averaging buffer of the radio. This method is only needed for averaging radios, such as IEEE 802.15.4 ones.
- `CTC_TIMER_NOW()`. Returns the current value of the accurate, free-running timer used for synchronization.

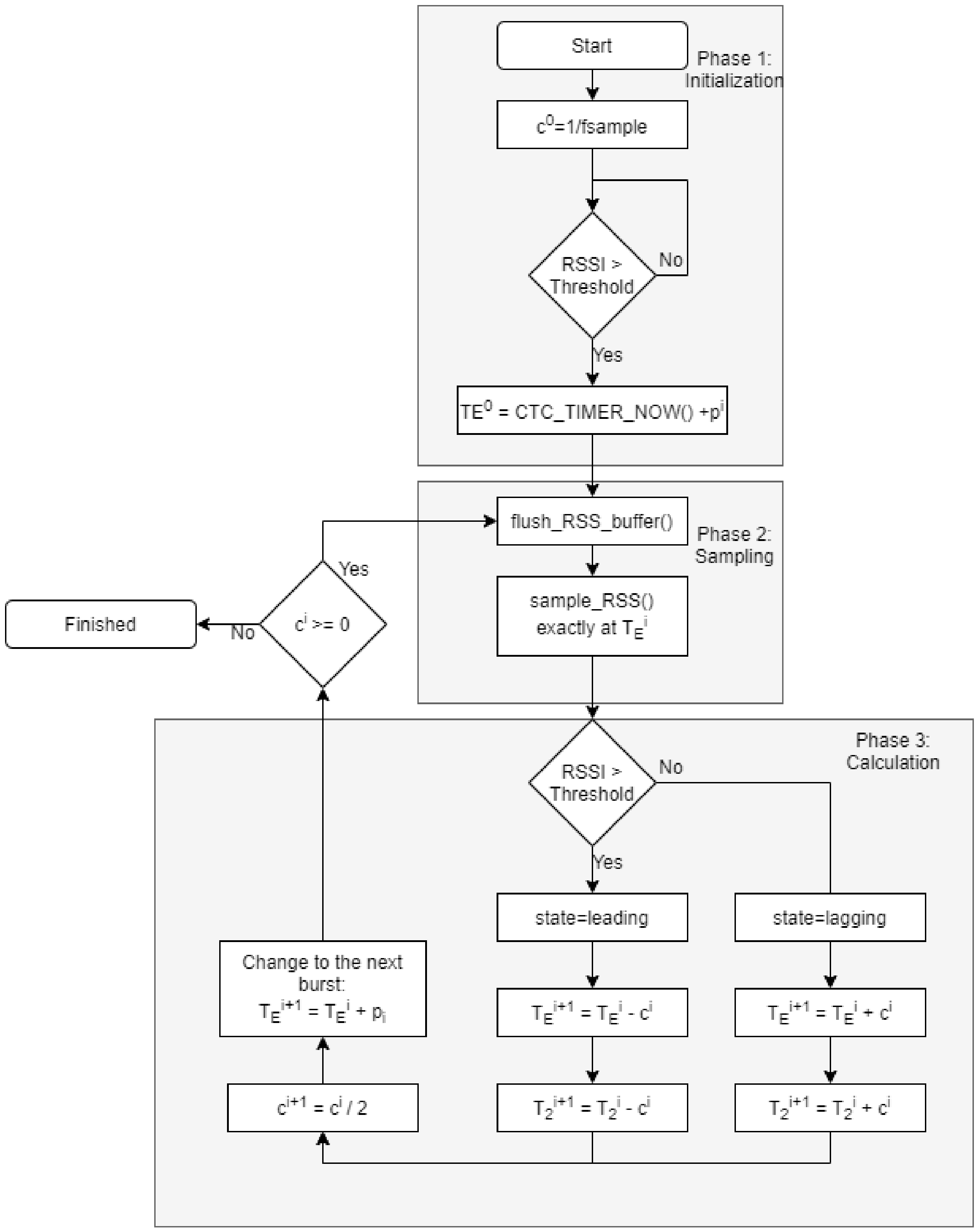


Figure 4.11: State machine for compensating the averaging delay ($T_{averaging}$) by using a binary search algorithm on the synchronization preamble. The timestamp T_2 will be corrected and can be used by the upper blocks, once the finished state is reached.

4.4 Validation/Correction Blocks

In this section, blocks for the validation and correction of the synchronized clock are grouped. Thereby, the equations for converting the transmitter's clock to the receiver's clock are given, as well as the implementation of the clock skew estimation and filtering algorithms explained in more detail in Section 3.5.

4.4.1 Clock Skew Estimation and Filtering Block

This block gets the synchronization tuple (T_1, T_2) as input parameter and generates the clock skew and offset values needed for the clock correction. The first step is to filter the synchronization tuple using RANSAC, as described in Section 3.5. Once the RANSAC algorithm filtered the outliers, the clock skew is estimated using linear regression. Because both algorithms are computationally intensive, the maximum amount of rounds can be adjusted. Likewise, lowering the number of stored synchronization tuples will help decreasing the consumed memory, but this influences the synchronization accuracy, which can be seen in Section 5. This block passes the result, consisting of the clock skew and the offset, to the upper blocks, once the calculation is finished.

4.4.2 Clock Correction Block

The input to this block is the skew and offset calculated in the clock skew estimation and filtering block. By using these two values, the receiver can estimate the clock of the transmitter. Assuming that the transmitting node is device A and the receiving node is device B, the following relationship holds:

$$T_B = \text{skew} * T_A + \text{offset} \tag{4.7}$$

Vice-versa, if T_A should be calculated, given T_B:

$$T_A = \frac{T_B - \text{offset}}{\text{skew}} \tag{4.8}$$

The clock correction block provides methods to convert timestamps between the two devices, instead of correcting the receiver's clock. Doing that would mean that the local clock needs to be stopped, set to a new value and started again. This process would decrease the achievable accuracy, as the starting delay of the clock is non-deterministic. The application can use the conversion functions at any time, once the clock correction block receives the first skew and offset parameter from the lower blocks.

5 Evaluation

In this section, we evaluate X-Sync in terms of synchronization accuracy, energy efficiency, memory footprint, reliability, and real-world usage. Therefore, we first start with defining the experimental setup in Section 5.1, used for all evaluations in this thesis. Next, the synchronization accuracy is evaluated (Section 5.2), which primarily depends on the interval between two synchronization messages I_{send}, the length of the synchronization preamble N_s and the number of stored synchronization pairs N. Following that, the energy consumption (Section 5.3) is calculated theoretically, which is needed for identifying parameters and adjusting them accordingly. Another very important aspect of X-Sync is the memory consumption (Section 5.4), which is calculated and compared for each hardware platform. Furthermore, the reliability of X-Sync is also evaluated (Section 5.5) and finally, a practical use of X-Sync is shown in a long-term, real-world scenario (Section 5.6).

5.1 Experimental Setup

In this section, we describe the experimental setup used for all evaluations. Figure 5.1 shows a schematic representation of the node arrangement, which consists of four wireless sensor nodes, equally spaced such that each X-Sync transmission path is exactly one meter. During the tests, nodes can be disabled, but the placement and the connections of all nodes remain unchanged.

The goal of this experimental setup is to measure the relative synchronization accuracy between the transmitting and the receiving node. In order to achieve that, the synchronized clock has to be sampled at the exact same time on both nodes. All hardware platforms had at least one free general purpose input output pin available, where a GPIO interrupt can be attached. Every time a node registers a rising edge on this pin, the synchronized clock signal is sampled. This sampling signal is connected to each node and clocked at f_{sample}, which generates a rising edge GPIO interrupt. A function generator generates this signal on synchronized outputs and it gets distributed using short BNC cables to lower cross-talk and noise. When a rising edge event is detected, the nodes immediately sample and transmit the synchronized timestamp using an UART connection to a database hosted on an attached PC.

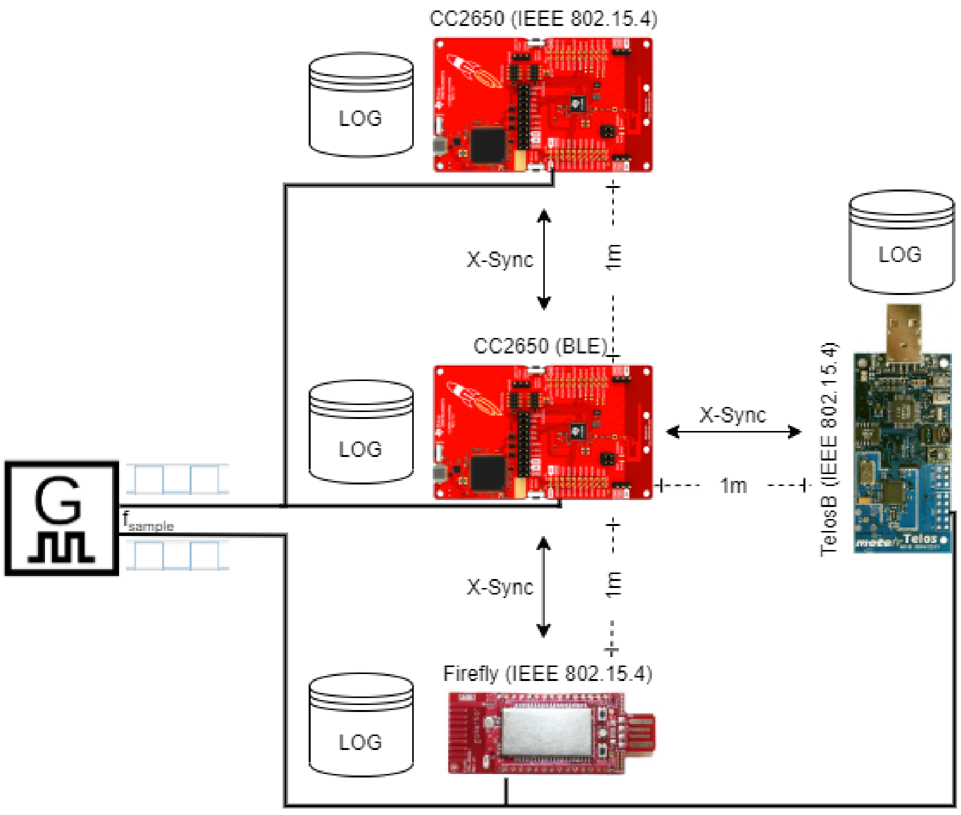


Figure 5.1: The experimental setup used for all evaluations showing the arrangement of the wireless sensor nodes. Each node is connected to a synchronized signal using a free GPIO pin. Whenever a rising edge is detected, the synchronized clock is sampled and stored to a database using an UART connection.

X-Sync follows a highly configurable approach, where parameters can be set depending on the application needs. The following sections describe the used settings in more detail.

5.1.1 Parameters

The following parameters will be changed and varied throughout the evaluations in order to show the impact on synchronization accuracy, energy consumption, reliability and memory footprint:

- **Synchronization Interval (I_{send}):** The time between two consecutive X-Sync synchronization messages.
- **Number of Synchronization Preamble Bursts (N_s):** The number of bursts of the synchronization preamble N_s that is used for compensating the timestamp T_2.
- **Number of Stored Synchronization Pairs (N):** The number of stored synchronization pairs (T_1,T_2) used to estimate the clock skew and the offset.

Additionally, the following evaluation parameters are defined:

- **Sampling Frequency (f_{sample}):** The rectangular-shaped sampling signal is connected to each node with a frequency of f_{sample}. Whenever a rising edge on this line is detected at any node, a GPIO interrupt is triggered and the synchronized clock is sampled immediately. This process interrupts any ongoing transmission/reception for a short amount of time. Therefore, the sampling frequency must not be too high as it would significantly lower the synchronization accuracy. Furthermore, sampling at a too low frequency might not lead to correct results. Hence, a sampling clock frequency of 1 Hz was chosen as a compromise.
- **Test Duration:** Choosing a test duration which is too short results in an obfuscated reality, where the results might be better than expected. Likewise, setting the test duration too long means very long test runs, which do not add any value to the results.
- **Test Repetitions:** Having independent runs lowers the environmental effects which are likely to occur, such as RF disturbances on the shared CTC channel. Thus repeating the measurement is necessary.

5.1.2 X-Sync Message

The parts included in a X-Sync message are highly configurable and can be tailored for a specific application. For this evaluation, the X-Sync message parts were chosen to be memory and energy efficient, as well as reliable:

- **CTC Preamble:** The preamble is straightly taken from X-Burst [20], in order to maintain backwards compatibility, with a few adjustments. This preamble is five bursts long and defined as follows: [$192\mu s$, $256\mu s$, $192\mu s$, $192\mu s$, $192\mu s$]. The time between two consecutive energy bursts in the synchronization preamble is chosen such that even the slowest node, i.e., the TelosB, has finished a full iteration of the synchronization state machine.
- **Synchronization Preamble:** In the following evaluations, X-Sync uses the shortest possible burst length ($192\mu s$ for each energy burst in the synchronization preamble.) repeated N_s times. This preamble is used to compensate for the delays inherent in packet-based CTC communications which results in T_2 being corrected accordingly.
- **CTC Header:** The one-byte CTC header defines the included optional data. For this evaluations, only the synchronization message, the payload length and the checksum bits are set, which are described in more detail below.
- **Timestamp T_1:** To allow fast clock rates, a 64-bit timestamp was chosen. This allows even the CC2650 (48 MHz) to run continuously without overflows.
- **Length Byte:** To further increase the data integrity, the packet length is added to each message, which gets verified upon reception.
- **Checksum:** A checksum is added to each X-Sync transmission to increase the data integrity and detect bit errors.

All other optional fields, including the payload have been omitted.

5.1.3 Block Configuration

The blocks of X-Sync can be configured individually, which includes the following settings:

- **Coding Scheme:** A reliable 2-bit encoding is chosen such that the reliability of the X-Sync is increased. The payload is omitted to lower the number of energy bursts and therefore increasing energy efficiency.
- **Communication Alphabet:** A reliable communication alphabet was chosen. As the transmission of the timestamp (T2) is not time-critical, using a more reliably alphabet at the cost of data rate is feasible.
- **Averaging Delay $\tau_{averaging}$ Compensation:** For each transmitter-receiver pair (i, j) exists exactly one averaging delay $\tau_{averaging\_i,j}$ in a static environment, which was measured at the beginning of each evaluation. Since we evaluate X-Sync in a static environment, as described in Section 3.4.1, this compensation factor can be measured and used to compensate the averaging delay on all receiving nodes.
- **CTC channel:** The radio channel has to be chosen such that there is a spectral overlap between the IEEE 802.15.4 and the BLE radio. For all IEEE 802.15.4 devices, channel 24 was chosen which overlaps with BLE channel 32. The channel mapping was already discussed and can be found in Section 2.1.

5.1.4 Measurement Procedure

Each evaluation consists of **two** measurements:

- **CC2650 (BLE) to all other nodes (IEEE 802.15.4).** In the first measurement, the CC2650 (BLE) node acts as transmitter, and all other nodes as receivers. The relative synchronization error is always measured pair-wise between the transmitter and the receiver.
- **All other nodes (IEEE 802.15.4) to the CC2650 (BLE) node.** In the second measurement, the CC2650 (BLE) node acts as receiver and all other nodes as transmitter in an alternating manner, i.e., only one transmitter is active during an evaluation. This was achieved by completely erasing the inactive nodes flash memory. These three measurements with different transmitters are then combined and plotted as one graph.

5.2 Synchronization Accuracy

The synchronization accuracy primarily depends on the following X-Sync parameters: the synchronization interval (I_{send}), the synchronization preamble length (N_s) and the number of stored synchronization pairs (N). Each of those parameters has implications on the memory footprint, the energy consumption and the reliability. The optimum of these parameters has to be found before X-Sync can be further evaluated. Thereby, the following parameters are used:

Parameter	Value
Synchronization Interval (I_{send})	1s, 10s, 30s, 60s, 180s, 300s
Number of Synchronization Preamble Bursts (N_s):	1, 4, 6, 8, 10, 12, 16, 20
Number of Stored Synchronization Pairs (N)	5, 8, 10, 20, 30, 40
External Sampling Clock Frequency (f_{sample})	1 Hz
Test Duration	$200 * I_{send}$
Test Repetitions	3

Table 5.1: X-Sync configuration parameters used to test the achievable synchronization accuracy. The synchronization interval (I_{send}), the number of synchronization preamble bursts (N_s) and the number of stored synchronization Pairs (N) get varied in every iteration of the following evaluations.

5.2.1 Synchronization Interval I_{send}

The synchronization accuracy primarily depends on the synchronization interval (I_{send}), i.e., on the duration between two consecutive X-Sync messages. Therefore, I_{send} is varied to prove its impact on the achievable accuracy and to find the optimum (if any). Higher I_{send} expectedly results in a higher energy consumption, while setting I_{send} too low results in a degraded synchronization accuracy. The other relevant parameters are set such that they do not limit the accuracy ($N_s = 20$, $N = 40$) and this tests were only repeated once because of the long test durations. Both directions BLE to IEEE 802.15.4 and vice-versa are evaluated in independent test runs:

BLE to IEEE 802.15.4

In this test, the CC2650 node in BLE mode acts as transmitter and reference clock, and all other nodes act as receivers. Hereby, N_s and N are configured such that these parameters do not influence the synchronization accuracy and the synchronization interval is varied between 1s and 300s. A CC2650 node (BLE) acts as the clock reference, which transmits X-Sync messages to all three receivers simultaneously in an interval of I_{send}, repeated in three independent runs in order to minimize external disturbances on the channel.

Figure 5.2 shows the results of the first evaluation. As expected, the higher the send interval I_{send}, the lower the synchronization accuracy and vice-versa. Furthermore, the $2.3\mu s$ accuracy goal specified in Section 3.1 was reached on all nodes except for the TelosB, because of constrained hardware capabilities. Especially the lack of an external crystal, the slow clock speed of 3.9 MHz and the lack of pre-emptive interrupts limit the achievable accuracy. Nevertheless, the TelosB node stayed at around $100\mu s$ relative synchronization accuracy, which is still a satisfactory result for this node. The results can be seen in more detail in Table 5.2, which shows the statistics of the relative synchronization error between the transmitter and the receiver. This includes the maximum (MAX), the minimum (MIN), the average (AVG) and the standard deviation (STD) of the dataset. Regarding the optimum value for I_{send} between energy consumption and accuracy, it can be seen that the synchronization accuracy obeys a linear behaviour to about 60s. In real world applications, a synchronization interval of 60s is reasonable and can therefore be seen as an optimum. For very high values of I_{send}, the clock drift is becoming visible, which means that the receiving node is unable to compensate for the clock drift, or that the clock is drifting in a non-linear fashion because of second order effects. These effects could theoretically be compensated by using higher-order, non-linear clock skew estimation algorithms [62], with the downside of needing processing power.

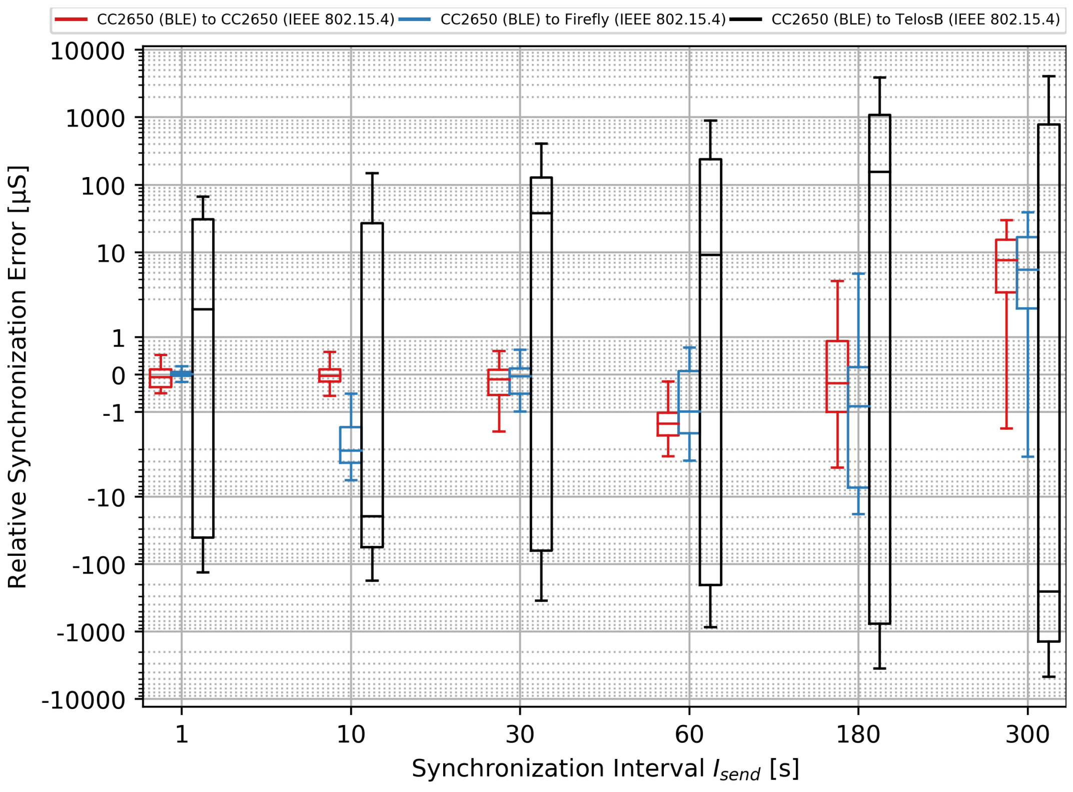


Figure 5.2: Evaluation of the synchronization accuracy depending on the synchronization interval I_{send}. A CC2650 node in BLE mode was used as transmitter and all other nodes were configured as receivers. The results are shown as a boxplot diagram, where the X-axis shows the synchronization interval and the Y-axis shows the relative synchronization error between transmitter and receiver.

IEEE 802.15.4 to BLE

Figure 5.3 shows the second evaluation regarding the synchronization accuracy. Each IEEE 802.15.4 node acts as transmitter once, while the CC2650 node in BLE mode acts as receiver during the whole test. For each test, two nodes are always active, with all other nodes being disabled. Only when the TelosB is used as transmitter, the goal of $2.3\mu s$ could not be reached. Nevertheless, the relative synchronization error stays below $30\mu s$ on the TelosB, which is still very good. The reason for the worse performance is its

Receiver	I_{send} [s]	MAX [μs]	MIN [μs]	AVG [μs]	STD [μs]
	1	0.517	-0.504	-0.072	0.281
	10	0.601	-0.566	-0.029	0.231
CC2650	30	0.621	-2.108	-0.275	0.536
(IEEE 802.15.4)	60	-0.191	-2.566	-1.351	0.435
	180	5.080	-5.629	0.159	1.940
	300	73.517	-1.441	11.421	13.219
	1	0.219	-0.218	0.006	0.094
	10	1.511	-6.156	-2.457	1.396
Firefly	30	0.657	-0.989	-0.152	0.395
(IEEE 802.15.4)	60	2.615	-2.947	-0.846	0.919
	180	4.823	-26.760	-4.189	6.592
	300	92.448	-2.552	12.061	17.648
	1	404.224	-132.109	12.707	93.454
	10	228.495	-175.839	-10.665	67.621
TelosB	30	457.828	-587.776	26.621	147.946
(IEEE 802.15.4)	60	3164.849	-3344.797	79.268	816.672
	180	6175.786	-11434.172	172.272	2097.065
	300	14263.370	-13516.214	-487.450	2975.600

Table 5.2: This table shows the results from Figure 5.2 in more detail. For each receiver, the maximum, the minimum, the average, and the standard deviation of the relative synchronization error are shown.

already discussed limited hardware capabilities. The average synchronization accuracy is not within this limit, but it could be drastically improved by using a simple round-trip synchronization algorithm on top of X-Sync. Regarding the optimum value for I_{send} between energy consumption and accuracy, it can be seen that the synchronization accuracy also obeys a linear behaviour to about 60s. The performance is degraded when using a high send interval I_{send} which is due to the assumption of a linear clock skew. All results can be seen in more detail in Table 5.3, which includes the maximum, the minimum, the average and the standard deviation of the relative synchronization accuracy.

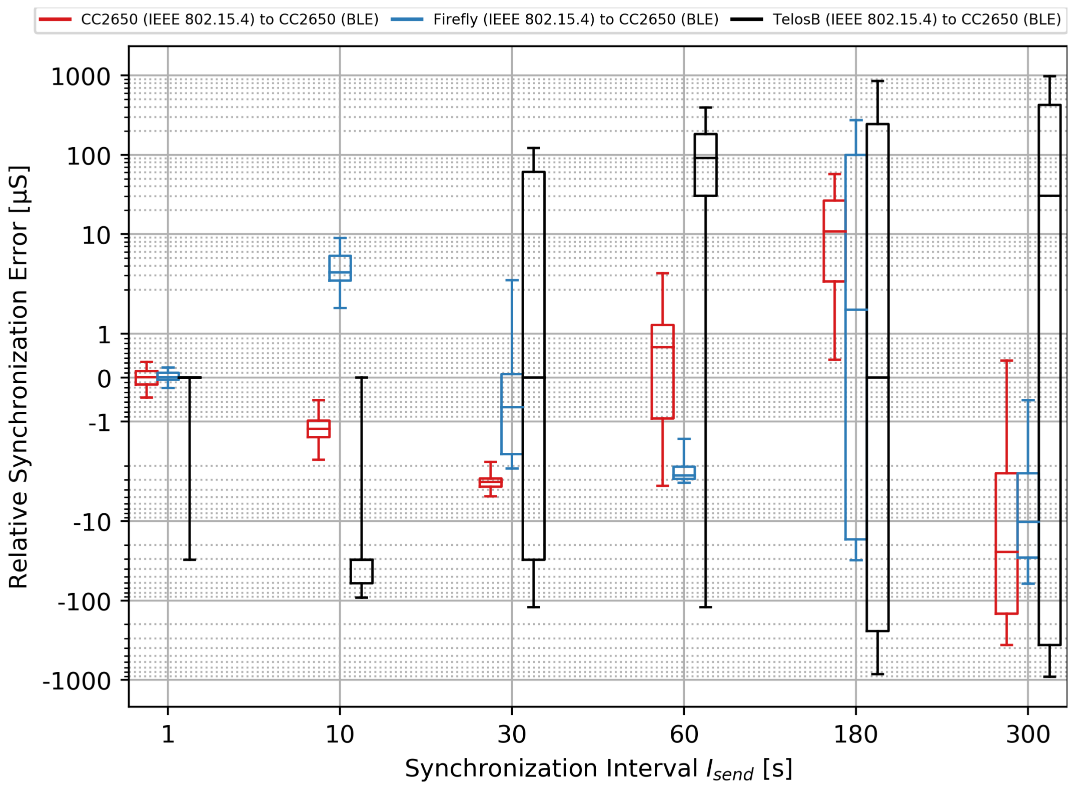


Figure 5.3: Evaluation of the synchronization accuracy depending on the synchronization interval I_{send}. The CC2650 node in BLE mode was configured as receiver, while the transmitter was changed throughout the test. A boxplot diagram shows the results, where the X-axis shows the synchronization interval and the Y-axis shows the relative synchronization error between transmitter and receiver.

5.2.2 Synchronization Preamble Length N_s

Another important parameter to evaluate is the synchronization preamble length N_s. A higher value results in a higher energy consumption per transmission, but may also result in a higher achievable accuracy. Therefore, N_s is varied to prove its impact on the achievable accuracy and to find the optimum (if any). The other relevant parameters are set such that they do not limit the accuracy ($I_{send} = 1s$, $N = 40$) and these tests were repeated three times.

Transmitter	I_{send} [s]	MAX [μs]	MIN [μs]	AVG [μs]	STD [μs]
	1	0.360	-0.452	-0.006	0.192
	10	-0.515	-1.869	-1.163	0.252
CC2650	30	-1.910	-5.390	-3.318	0.601
(IEEE 802.15.4)	60	3.194	-16.265	-0.713	4.039
	180	79.944	0.402	17.636	18.914
	300	0.381	-427.494	-91.692	127.408
	1	0.233	-0.298	0.019	0.120
	10	8.921	1.577	3.943	1.730
Firefly	30	3.077	-5.329	-0.731	1.045
(IEEE 802.15.4)	60	-1.392	-6.173	-2.488	0.498
	180	299.671	-31.017	49.759	90.209
	300	-0.517	-61.767	-16.695	17.648
	1	0.000	-30.518	-1.017	5.478
	10	0.000	-122.070	-40.843	24.614
TelosB	30	152.588	-122.070	9.087	59.847
(IEEE 802.15.4)	60	457.764	-152.588	111.279	119.306
	180	976.562	-1007.080	-9.588	333.246
	300	1037.598	-976.562	30.930	510.746

Table 5.3: This table shows the results from Figure 5.3 in more detail. For each transmitter, the maximum, the minimum, the average, and the standard deviation of the relative synchronization error are reported.

BLE to IEEE 802.15.4

Figure 5.4 shows such an evaluation where a CC2650 node in BLE mode sends a periodic synchronization message to all other nodes. The number of synchronization bursts gets varied between 1 and 20, and all other parameters are chosen such that they do not influence the result. Figure 5.4 and Table 5.4 show the results, which include the maximum (MAX), the minimum (MIN), the average (AVG) and the standard deviation (STD) of the relative synchronization error between transmitter and receiver. Having a synchronization preamble length of one cancels the binary search algorithm used for the averaging delay correction after the first iteration. Therefore, the estimated T_E is leading before the actual start of the energy burst T_R. This estimated T_E leads to a negative offset and a high standard deviation in Figure 5.4. Furthermore, setting N_s to four leaves the state machine in a lagging state where $T_E > T_R$, which can be seen as a positive offset in this graph. The state machine is still not finished, and therefore, the standard deviation is far off. Figure 5.4 furthermore shows that the synchronization state machine finishes if N_s is set to twelve or higher. Having a higher N_s does not significantly increase the relative synchronization error, as the state machine has already finished. The relative synchronization error of the TelosB node is, again, limited by its hardware capabilities, as already discussed in Section 5.2.

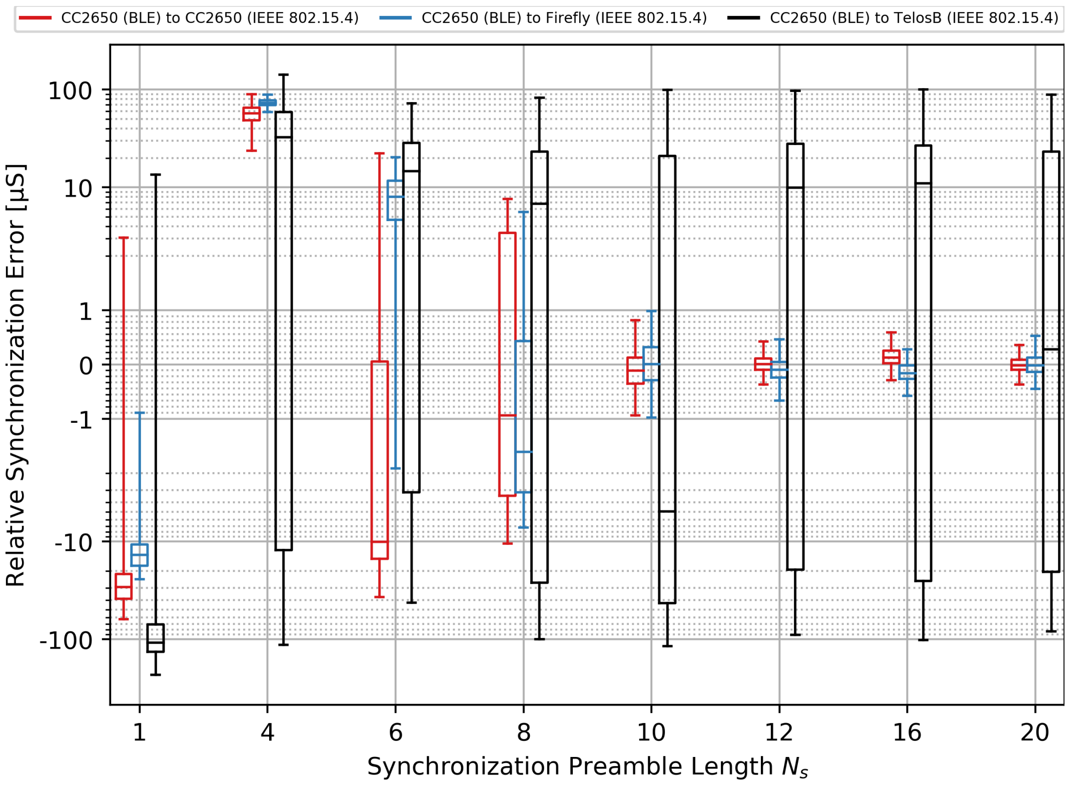


Figure 5.4: Evaluation of the synchronization accuracy in dependence of the synchronization preamble length N_s. Hereby, a CC2650 node in BLE mode was used as transmitter and all other nodes were configured as receivers. The result is shown as a boxplot diagram where the X-axis shows N_s and the Y-axis shows the relative synchronization error between the transmitter and the receiver.

Receiver	N_s	MAX [μs]	MIN [μs]	AVG [μs]	STD [μs]
	1	51.023	-111.373	-30.111	13.613
	4	141.335	-14.706	56.027	15.706
	6	54.481	-36.977	-7.586	10.985
CC2650	8	7.585	-10.581	-0.301	3.718
(IEEE 802.15.4))	10	1.252	-0.935	-0.091	0.382
	12	0.648	-0.373	0.006	0.158
	16	0.606	-0.290	0.140	0.158
	20	0.648	-0.394	-0.000	0.144
	1	2.628	-24.455	-13.923	4.807
	4	88.461	58.836	73.488	5.130
	6	20.253	-1.914	8.212	4.571
Firefly	8	6.441	-7.226	-1.315	2.440
(IEEE 802.15.4)	10	4.607	-0.976	0.030	0.491
	12	1.316	-0.705	-0.092	0.223
	16	1.566	-0.580	-0.149	0.185
	20	0.711	-0.455	-0.000	0.200
	1	176.821	-301.721	-105.105	48.967
	4	141.883	-379.554	20.141	62.181
	6	103.196	-173.554	5.979	39.810
TelosB	8	82.654	-174.659	-3.145	39.561
(IEEE 802.15.4)	10	98.967	-195.742	-11.570	44.112
	12	112.988	-163.887	-0.122	41.629
	16	1294.863	-202.637	2.688	100.517
	20	143.738	-225.178	-1.501	49.896

Table 5.4: This table shows the results from Figure 5.4 in more detail. For each receiver, the maximum, the minimum, the average, and the standard deviation of the relative synchronization error are shown.

IEEE 802.15.4 to BLE

For the second evaluation, receivers and transmitters are switched and operate in unicast mode. Figure 5.5 shows the result of this evaluations and Table 5.5 shows the results in more detail, which includes the maximum, the minimum, the average and the standard deviation of the realative synchronization error between transmitter and receiver. Again, both leading and lagging states from the binary search algorithm can be seen as positive and negative offsets, respectively. The result is an alternating behaviour between detecting the burst too early and too late until the number of bursts is sufficient to finish the state machine.

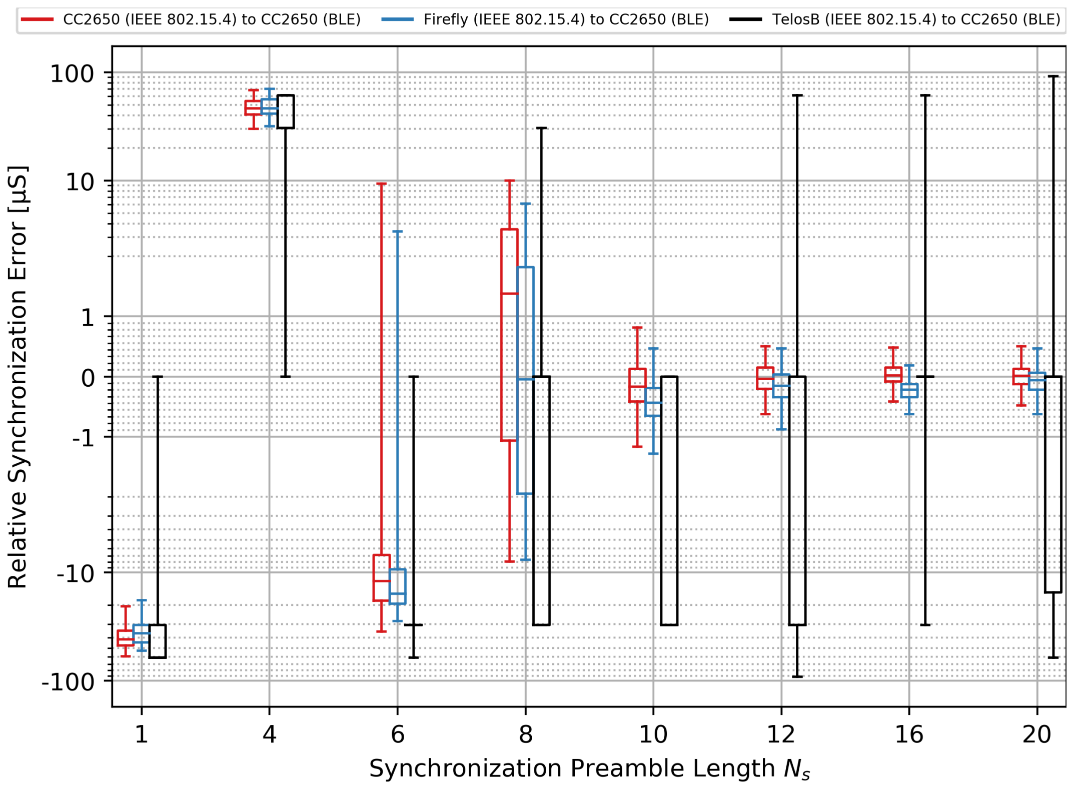


Figure 5.5: Evaluation of the synchronization accuracy depending on the synchronization preamble length N_s. Hereby a CC2650 node in BLE mode was used as receiver and all other nodes were one-by-one configured as transmitters. The results are shown as a boxplot diagram, where the X-axis shows N_s and the Y-axis shows the relative synchronization error between transmitter and receiver.

5.2.3 Number of Stored Synchronization Pairs N

The last parameter to evaluate is the number of stored synchronization pairs N. We expect that having a high N results in a high insensitivity against disturbances on the shared channel. On the other hand, the memory on embedded devices is limited and, thus, a optimum could be found (if any). The other relevant parameters are set such that they do not limit the accuracy ($I_{send} = 1s$, $N_s = 20$) and these tests were repeated three times.

Transmitter	N_{pre}	MAX [μs]	MIN [μs]	AVG [μs]	STD [μs]
	1	-20.458	-59.375	-40.619	8.050
	4	68.125	29.854	47.349	8.043
	6	22.229	-35.708	-11.501	10.309
CC2650	8	10.979	-7.917	1.142	3.413
(IEEE 802.15.4)	10	1.063	-1.167	-0.140	0.382
	12	0.500	-0.625	-0.037	0.236
	16	0.583	-0.542	0.040	0.180
	20	0.583	-0.479	0.022	0.198
	1	-18.063	-52.531	-37.049	7.639
	4	70.281	31.562	48.440	8.195
	6	22.750	-28.125	-13.894	7.955
Firefly	8	6.125	-10.125	-0.199	2.740
(IEEE 802.15.4)	10	0.594	-1.281	-0.415	0.354
	12	0.469	-1.000	-0.168	0.280
	16	0.312	-0.688	-0.230	0.170
	20	0.469	-0.719	-0.080	0.204
	1	0.000	-152.588	-48.014	17.298
	4	91.553	0.000	40.385	16.494
	6	0.000	-61.035	-26.652	11.844
TelosB	8	30.518	-91.553	-13.123	18.691
(IEEE 802.15.4)	10	0.000	-91.553	-22.888	14.341
	12	122.070	-122.070	-19.124	30.323
	16	61.035	-61.035	4.985	14.095
	20	152.588	-122.070	-6.816	24.669

Table 5.5: This table shows the results from Figure 5.5 in more detail. For each transmitter, the maximum, the minimum, the average, and the standard deviation of the relative synchronization error are shown.

BLE to IEEE 802.15.4

For the first test regarding the number of synchronization pairs N, the CC2650 node in BLE mode was configured as a transmitter, while all other nodes were configured as receivers. The parameters were chosen such that they do not influence this evaluation, and N was varied between 5 and 40. Figure 5.6 and Table 5.6 show the impact of the amount of stored synchronization pairs N to the relative synchronization accuracy. Having a higher N reduces the relative synchronization error on all nodes, except on the TelosB. On that platform, no high-frequency crystal oscillator is available which could be used as an internal clock reference. Instead, the high frequency clock source makes use of an internal RC oscillator which is highly unstable and non-linear. Because of the aforementioned reasons, the low-frequency real-time reference clock of 32kHz is used on the TelosB. While this clock inhibits very good linear characteristics, the frequency is very low and thus the binary search algorithm is only partly working. It simply does not achieve the clock granularity needed to decide whether the state machine is currently leading or lagging and therefore, a higher number of stored synchronization pairs does not help.

The other two nodes show that, albeit having a linear relationship, the number of synchronization pairs N does not have a big influence on the synchronization accuracy. This basically means that the filtering and estimation algorithms already work quite well with only a small number of stored samples. We chose N=20 to be a good compromise between accuracy and memory usage for all following evaluations.

Receiver	N	MAX [μs]	MIN [μs]	AVG [μs]	STD [μs]
	5	31.585	-24.623	-0.035	2.901
	8	96.190	-25.352	0.364	6.036
CC2650	10	129.565	-34.435	0.403	8.272
(IEEE 802.15.4)	20	2.106	-0.560	0.021	0.232
	30	0.815	-0.540	-0.023	0.175
	40	0.481	-0.435	-0.006	0.149
	5	1.691	-1.351	-0.023	0.437
	8	1.232	-1.976	-0.018	0.385
Firefly	10	0.961	-2.934	-0.007	0.355
(IEEE 802.15.4)	20	0.711	-0.851	-0.034	0.233
	30	0.461	-0.601	-0.069	0.197
	40	0.503	-0.601	-0.015	0.184
	5	225.321	-508.617	-25.834	98.438
	8	163.071	-140.117	-8.160	36.156
TelosB	10	266.550	-298.971	-16.367	59.163
(IEEE 802.15.4)	20	398.966	-705.138	-19.304	80.420
	30	211.487	-230.659	0.390	48.066
	40	152.008	-200.825	-20.582	43.935

Table 5.6: This table shows the results from Figure 5.6 in more detail. For each receiver, the maximum, the minimum, the average, and the standard deviation of the relative synchronization error are shown.

IEEE 802.15.4 to BLE

This evaluation was repeated with the CC2650 BLE node configured as receiver and all other nodes configured as transmitters. Table 5.7 and Figure 5.7 show the results of this evaluation in detail, which include the maximum, the minimum, the average, and the standard deviation of the relative synchronization error between transmitter and receiver. Both show that the synchronization accuracy increases with increasing N, also seen in the first test of this section. As for the TelosB, it operates on a clock frequency of 32kHz, which means that the receiver also estimates the synchronized clock with a granularity of 32kHz. This results in distinct steps of approximately 30μs which are visible in the figure. For the other two nodes, it can again be seen that N does not have a big influence to the synchronization accuracy and a value of 20 is sufficient to reach the requested accuracy goal of 2.3μs.

Transmitter	N	MAX [μs]	MIN [μs]	AVG [μs]	STD [μs]
	5	2.438	-1.563	0.041	0.532
	8	1.250	-1.083	0.039	0.382
CC2650	10	1.563	-1.021	-0.029	0.335
(IEEE 802.15.4)	20	0.708	-0.771	-0.011	0.264
	30	0.646	-0.854	-0.021	0.234
	40	0.583	-0.583	0.006	0.203
	5	1.406	-2.063	-0.211	0.499
	8	1.156	-1.375	-0.194	0.414
Firefly	10	0.844	-1.469	-0.189	0.385
(IEEE 802.15.4)	20	0.687	-1.000	-0.187	0.269
	30	0.344	-0.813	-0.183	0.205
	40	0.437	-2.500	-0.140	0.224
	5	183.105	-122.070	-0.254	16.385
	8	762.939	-274.658	-6.307	54.025
TelosB	10	30.518	-61.035	-1.221	9.573
(IEEE 802.15.4)	20	30.518	-61.035	-0.814	8.772
	30	30.518	-61.035	0.051	8.170
	40	30.518	-91.553	-18.921	16.118

Table 5.7: This table shows the results from Figure 5.7 in more detail. For each transmitter, the maximum, the minimum, the average, and the standard deviation of the relative synchronization error is shown.

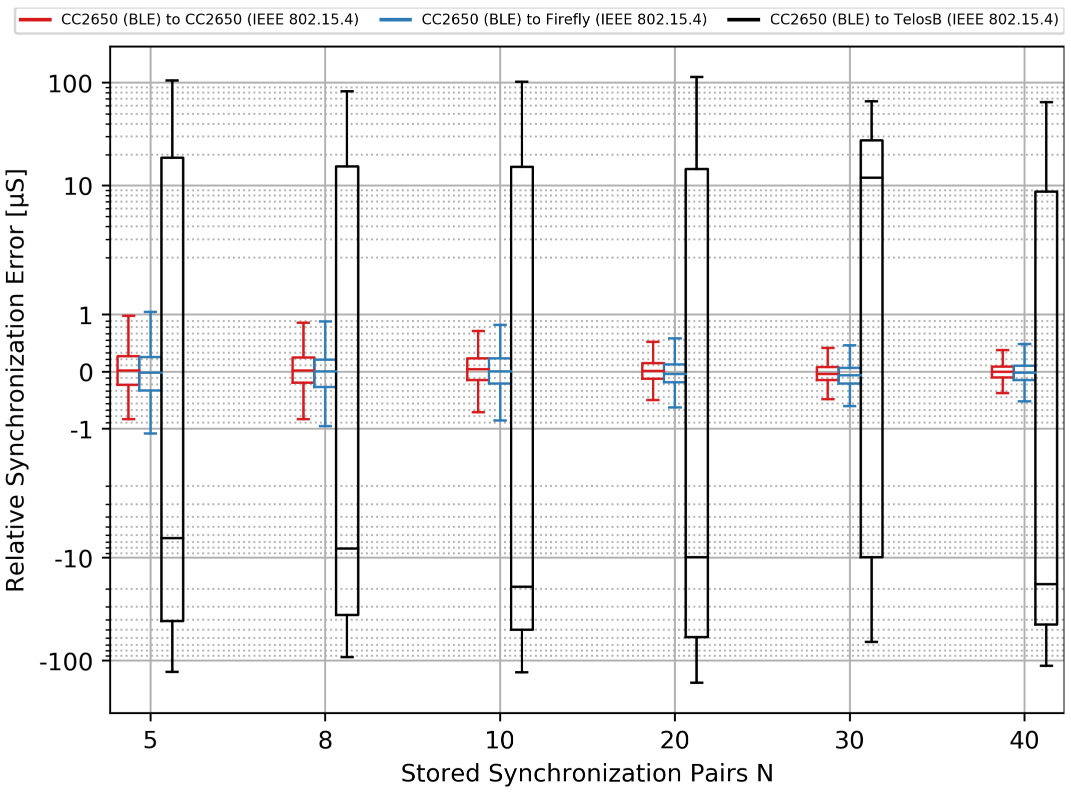


Figure 5.6: Evaluation of the synchronization accuracy depending on the number of stored synchronization pairs *N*. Hereby a CC2650 node in BLE mode was used as transmitter and all other nodes were configured as receivers. The results are shown as a boxplot diagram where the X-axis shows the amount of stored synchronization pairs *N* and the Y-axis shows the relative synchronization error between transmitter and receiver.

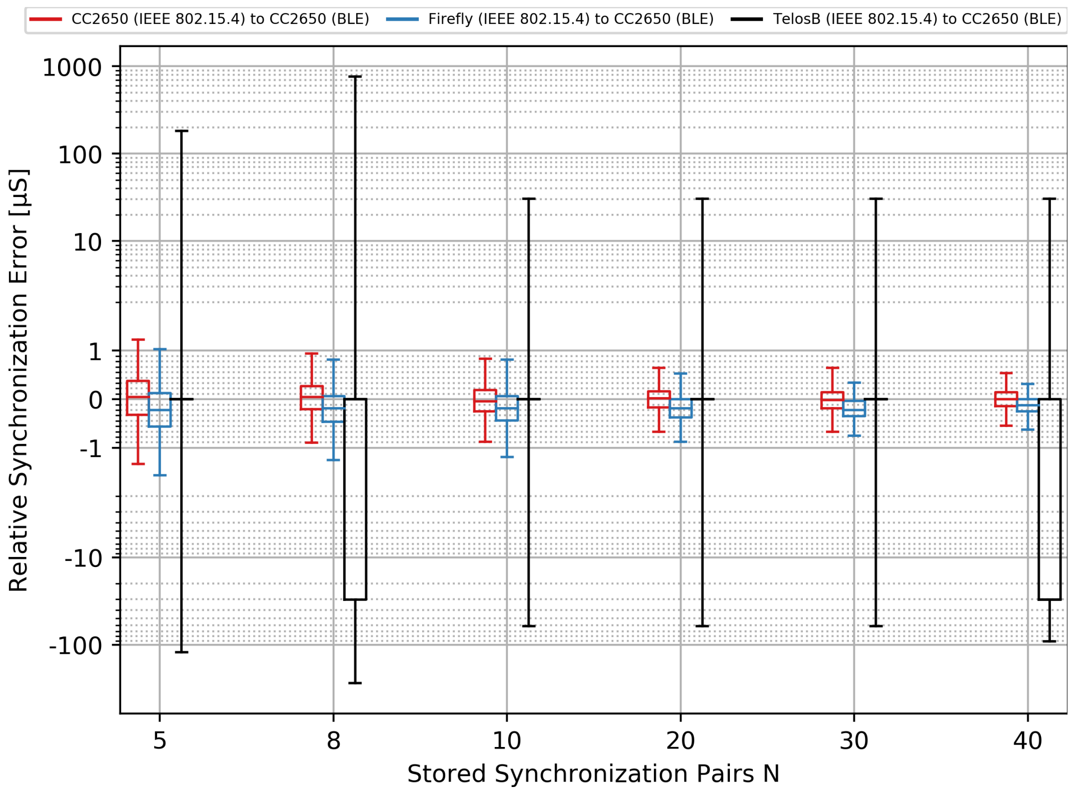


Figure 5.7: Evaluation of the synchronization accuracy depending on the number of stored synchronization pairs N. Hereby a CC2650 node in BLE mode was used as a receiver and all other nodes were configured one-by-one as transmitters. The results are shown as a boxplot diagram where the X-axis shows the amount of stored synchronization pairs N and the Y-axis shows the relative synchronization error between transmitter and receiver.

5.3 Energy Consumption

In Section 5.2, we showed the influence of the interval between synchronization messages I_{send} and the achievable synchronization accuracy. The lower I_{send}, the more X-Sync messages are sent, which has a direct impact on the energy consumption. The authors of [20] show that the energy consumption for the reception of CTC messages is determined by the duration of the RSS sampling window (reception window), while the energy consumption in case of transmissions mainly depends on the duration in which the radio is active, i.e, transmitting data packets. Each node has a specific current consumption during the transmission of energy bursts, the transmission of gaps (node is still active) and the reception per second (I_{active}, I_{TX} and I_{RX} respectively). This property can be taken from each node's datasheet ([23], [24] and [53]) and is summarized in Table 5.8. For simplicity, an active CPU is assumed during transmission and reception.

Mode	CC2650 [mA]	Firefly [mA]	TelosB [mA]
I_{active}	2.93	13	1.8
$I_{active+TX}$ @ 0dBm	9.03	24	19.5
$I_{active+RX}$	8.83	20	21.8

Table 5.8: Current consumption for each hardware platform in different operating modes (I_{active}, $I_{active+TX}$ and $I_{active+RX}$). This properties are taken from each node's datasheet ([23], [24] and [53]).

Assuming a nominal supply voltage of 3.3V for each hardware platform, the power consumption can be estimated as shown in Table 5.9.

Mode	CC2650 [mW]	Firefly [mW]	TelosB [mW]
P_{active}	9.67	42.9	5.94
$P_{active+TX}$ @ 0dBm	29.8	79.2	64.35
$P_{active+RX}$	29.14	66	71.94

Table 5.9: Power consumption for each hardware platform in different operating modes (P_{active}, $P_{active+TX}$ and $P_{active+RX}$).

The energy consumption of X-Sync transmissions and receptions strongly depends on the configuration, the alphabet, the coding scheme, and the chosen encoding. Hence, a concrete value about the additional energy consumption of a device using X-Sync cannot be given. However, to better understand the additional energy consumption caused by X-Sync, the consumption of a minimal synchronization message is shown. The optimal configuration value for N_s is hereby taken from the previous evaluation (Section 5.2) and is set to 12. The example given uses the following configuration:

- Alphabet: The used alphabet is given in Table 4.2.
- Encoding: 2-bit, burst-only encoding, i.e., each duration contains 2 bit of information, taken from Table 4.3.
- Average burst duration for random data: Using Table 4.3 the average burst duration can be calculated which results to be $240\mu s$.
- Timestamp Size: 64 bit.
- Time between two consecutive energy burst (typical): $200\mu s$.
- Number of synchronization preamble bursts $N_s = 12$.

Using this parameters, Table 5.10 shows a simple example of a minimal X-Sync message consisting of a CTC preamble, a synchronization preamble, a CTC header, and the timestamp T_1. While adding a checksum is not needed, it ensures data integrity. The table shows the average time spend in each operating mode during reception ($t_{active} + t_{RX}$) and transmission (t_{TX}).

Part of X-Sync Message	Size [Bytes]	# Bursts	Transmission		Reception
			t_{active} (AVG) [μs]	t_{TX} (AVG) [μs]	t_{RX} (AVG) [μs]
CTC Preamble	-	5	1000	1024	2024
Synchronization Preamble	-	$N_s = 12$	2400	2304	4704
CTC Header	1	4	800	960	1760
Timestamp	8	32	6400	7680	14080
Checksum	1	4	800	960	1760
Result	-	57	11400	12928	24328

Table 5.10: The specific parts of a minimal X-Sync message, the amount of bursts needed for the transmission/reception and the average time spent in each operating mode.

Using the results from Table 5.10, the average power usage per transmission/reception of this X-Sync message can be calculated. The results can be seen in Table 5.11 which shows that depending on the hardware platform, the power consumption can vary greatly and transmitting the least amount of X-Sync message should always be the goal. Nevertheless, the energy consumption per X-Sync transmission is more than acceptable.

Node	Energy Usage per Transmission [mWs] ($t_{active} * P_{active} + t_{TX} * P_{active+TX}$)	Energy Usage per Reception [mWs] ($t_{RX} * P_{active+RX}$)
CC2650	495.49	708.92
Firefly	1512.96	1605.65
TelosB	899.93	1750.16

Table 5.11: Energy usage for each hardware platform for transmission and reception of a minimal X-Sync message.

5.4 Memory Footprint

IoT devices are limited in terms of hardware capabilities, such as processing speed, power consumption, and available memory. In this section, we evaluate the static memory usage that X-Sync adds to an existing IoT application. Table 5.12 lists a typical X-Sync configuration, which is used as an example to calculate the memory consumption. Hereby, the results from the previous evaluations (Section 5.2) were used, which define the synchronization preamble length $N_s = 12$ and the number of stored synchronization pairs $N = 20$.

Parameter	Value
Number of Synchronization Preamble Bursts (N_s):	12
Number of stored Synchronization Pairs (N)	20
Size of Timestamp T_2	64 Bit

Table 5.12: X-Sync configuration parameters used to evaluate the memory footprint. All parameters not relevant for the memory footprint are omitted.

Depending on the used node, the memory footprint varies as different compilers, architectures, and HAL layers are used. Table 5.13 shows the ROM and RAM consumption of X-Sync for each platform. The ROM usage is roughly equal on the different platforms while the RAM usage differs significantly. Thereby, the TelosB node does not support radio queues, which explains the low memory footprint of 1.18 kB. Both, the Firefly and the CC2650 node make use of radio queues, whose structure and size is platform dependent. The Firefly node allows to specify such a queue in an more efficient way, which explains the rather low RAM usage, despite the use of a queue. In comparison to X-Burst, X-Sync undeniably needs more resources. That can be explained by the use of 64-bit timestamps for all timing-related functions, radio queues, RANSAC filtering and the added state machines for processing the synchronization preambles.

	X-Sync		X-Burst	
Node	ROM usage [kB]	RAM usage [kB]	ROM usage	RAM usage
CC2650 (BLE)	15.20	9.90	7.36	1.10
CC2650 (IEEE 802.15.4)	15.44	9.97	7.86	1.17
Firefly (IEEE 802.15.4)	12.81	1.76	6.97	1.26
TelosB (IEEE 802.15.4)	14.20	1.18	6.78	0.75

Table 5.13: The static memory usage in terms of RAM and ROM when X-Sync is added to an existing IoT application using a typical configuration listed in Table 5.12. Additionally, the RAM and ROM usage of X-Burst are included, taken from [42].

5.5 Reliability

The reliability of X-Sync transmissions primarily depends on the reliability of the cross-technology communication scheme itself, which is used to transmit the timestamp. The authors of X-Burst evaluate the transmission reliability comprehensively, which can be found in their paper [42] and is thus not repeated in this evaluation. Receiving a correct message, which includes the timestamp T_1, does not mean that the sampling delay compensation algorithm is working reliably. Disturbances on the channel may lead to wrong binary search results, and thus, the resulting time synchronization pairs (T_1, T_2) need to be filtered and stored in a buffer, as described in Section 3.5. Having a higher number of stored synchronization pairs N will lead to a higher resistance against disturbances, but increases the memory consumption, and results in a higher CPU load. This is shown in Section 5.2.3. Due to the RANSAC filtering of the samples in conjunction with the linear regression, only a very limited number of pairs is needed to accomplish very good results. Thus, having a higher number of stored synchronization pairs N does not influence the result greatly and N=20 is used as a compromise for the following evaluation.

5.6 X-Sync in Action

The evaluations of the previous sections were used to determine the optimal values for X-Sync, regarding energy and memory consumption, reliability and accuracy. Table 5.14 summarizes the optional parameters. Using this parameters, we did a long-term (35h) evaluation of X-Sync. This was only done once because of the long test durations.

Parameter	Value
Synchronization Interval (I_{send})	60s
Number of Synchronization Preamble Bursts (N_s):	12
Number of stored Synchronization Pairs (N)	20

Table 5.14: X-Sync configuration parameters used for the long-term evaluation.

BLE to IEEE 802.15.4

In this evaluation, the CC2650 (BLE) node was configured as transmitter and all other nodes were configured as receivers. The results are shown in the time domain (Figure 5.8) and are summarized in Table 5.15. At t=1h and t=17h, a disturbance on the channel can be seen on both the CC2650 node in IEEE 802.15.4 mode and the Firefly node. On the TelosB, this disturbance is not significant enough to be visible, as the synchronization accuracy is limited by the hardware capabilities of the node. This is because the sampling delay $T_{sampling}$ correction algorithm needs a high-frequency and stable clock source in order to work properly. A few minor disturbances on the channel can also be seen throughout the evaluation (t=6h, t=22h, t=28h), but their impact is filtered by the use of RANSAC and thus the effect on the synchronization accuracy is only minor.

Figure 5.9 shows the accuracy between t=5h and t=6h in more detail to highlight the expected short-term variations. It can be seen that the performance of both the CC2650 node, as well as the Firefly node is remarkable and no major outliers can be seen. As for the TelosB, quantization effects can clearly be seen, which are caused by the fact that double-precision floating point arithmetic is not supported by the used GCC compiler. X-Sync therefore runs on single-point arithmetic that results in visible quantization steps.

Summarized, the achieved relative synchronization accuracy is outstanding ($< \pm 45 \mu s$ for the CC2650 and the Firefly) and still note-able on the TelosB ($< \pm 4ms$) given the hardware constraints. To further show the accuracy of X-Sync, the cumulative distribution function (CDF) of the modulus of the relative synchronization error was calculated for each transmitter and receiver combination and is shown in Figure 5.10. These results are further summarized in Table 5.16, which show the median (50%) as well as the 95% and 99% value of the CDF. It can be seen that the modulus of the relative synchronization accuracy stays below $3 \mu s$ on the CC2650 platform and below $7 \mu s$ on the Firefly for 95% of the time, which is remarkable. The TelosB results still show that the synchronization accuracy stays well below $1 \mu s$ for the same 95% of the time.

Receiver	MAX [μs]	MIN [μs]	AVG [μs]	STD [μs]
CC2650 (IEEE 802.15.4)	15.788	-13.754	0.320	1.556
Firefly (IEEE 802.15.4)	35.032	-44.239	0.552	3.897
TelosB (IEEE 802.15.4)	3745.703	-3774.859	-30.640	432.918

Table 5.15: This table shows the results of the long-term X-Sync evaluation (Figure 5.8) in more detail. For each receiver, the maximum, the minimum, the average, and the standard deviation of the relative synchronization error is shown.

Receiver	Median (50%)	95%	99%
CC2650 (IEEE 802.15.4)	0.580 μs	2.517 μs	7.233 μs
Firefly (IEEE 802.15.4)	0.927 μs	6.157 μs	17.989 μs
TelosB (IEEE 802.15.4)	246.771 μs	812.040 μs	1311.051 μs

Table 5.16: The summarized results of the CDF (Figure 5.10) showing the median (50%) as well as the 95% and 99% modulo of the relative synchronization accuracy.

IEEE 802.15.4 to BLE

The same test is repeated with the CC2650 BLE node configured as receiver and all other nodes, one-by-one, as transmitter. The results of this time-domain evaluation can be seen in Figure 5.11 and is summarized in Table 5.17. Because of the fact that the transmitting capabilities were evaluated one-by-one in a unicast configuration, the nodes do not see the disturbances on the channel at the same time. Some major disturbances occurred, whose impact was filtered using RANSAC and linear regression. Both, the CC2650 node in IEEE 802.15.4 mode, as well as the Firefly node stay below a relative synchronization error of $\pm 45\mu s$ even during those events. The TelosB node's relative synchronization accuracy is, again, limited by its hardware capabilities, which was already discussed in Section 5.2. The discrete steps can be clearly seen in the results, which show that the used timer on the TelosB is only clocking at 32kHz. Nevertheless, the TelosB achieves a respectable performance when used as X-Sync transmitter and always stays below $\pm 125\mu s$.

Figure 5.12 shows the accuracy between t=5h and t=6h in more detail to highlight the expected short-term variations. It can be seen that the performance of both the CC2650 node, as well as the Firefly node is remarkable and no major outliers can be seen. As for the TelosB, the performance is limited by the use of the low-frequency external 32kHz crystal oscillator, which is needed for stability and is the only crystal oscillator available on this hardware platform. Thus, one step is $\frac{1}{32768} = 30.517\mu s$ which is exactly what can be seen in the graph. Hereby, the TelosB nodes is configured as transmitter and thus, the low frequency of the used oscillator influences the results much less than in the other direction. This is because the sampling delay ($\tau_{sampling}$) correction algorithm needs a high-frequency and accurate clock source in order to work properly.

Summarized, the achieved relative synchronization accuracy is outstanding ($< \pm 100\mu s$ for the CC2650 and the Firefly) and still very good on the TelosB ($< \pm 125ms$) given it's hardware constraints. To further show the accuracy of X-Sync, the cumulative distribution function (CDF) of the modulus of the relative synchronization error was calculated for each transmitter and receiver combination and is shown in Figure 5.13. These results are further summarized in Table 5.18, which shows the median (50%) as well as the 95% and 99% value of the CDF. Again, a step in the distribution can be seen on the TelosB platform, caused by the use of the 32kHz crystal oscillator. Nevertheless, the modulus of the relative synchronization error stays below $31\mu s$ regardless of the used platform for 95% of the time.

Transmitter	MAX [μs]	MIN [μs]	AVG [μs]	STD [μs]
CC2650 (IEEE 802.15.4)	96.923	-29.160	-0.082	4.706
Firefly (IEEE 802.15.4)	51.827	-94.298	-1.589	8.771
TelosB (IEEE 802.15.4)	91.553	-122.070	-25.830	12.728

Table 5.17: This table shows the results of the long-term X-Sync evaluation (Figure 5.11) in more detail. For each transmitter, the maximum, the minimum, the average, and the standard deviation of the relative synchronization error is shown.

Transmitter	Median (50%)	95%	99%
CC2650 (IEEE 802.15.4)	0.819 μs	4.848 μs	13.598 μs
Firefly (IEEE 802.15.4)	1.767 μs	10.798 μs	48.017 μs
TelosB (IEEE 802.15.4)	30.518 μs	30.518 μs	61.035 μs

Table 5.18: The summarized results of the CDF (Figure 5.13) showing the median (50%) as well as the 95% and 99% modulo of the relative synchronization accuracy.

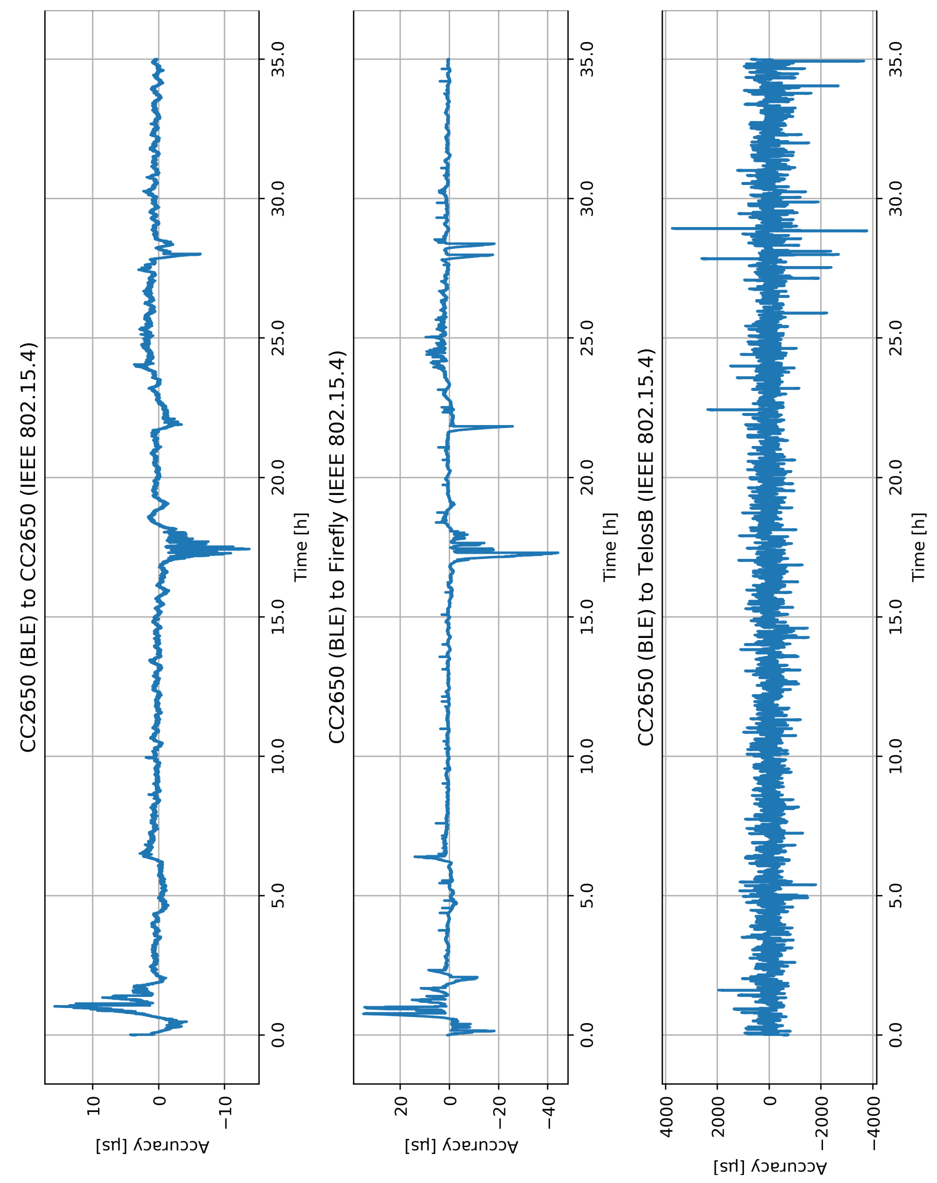


Figure 5.8: This figure shows the time-domain plot of the long-term X-Sync evaluation. Hereby the CC2650 node in BLE mode was used as a transmitter and all other nodes were configured as receivers, forming a broadcast configuration.

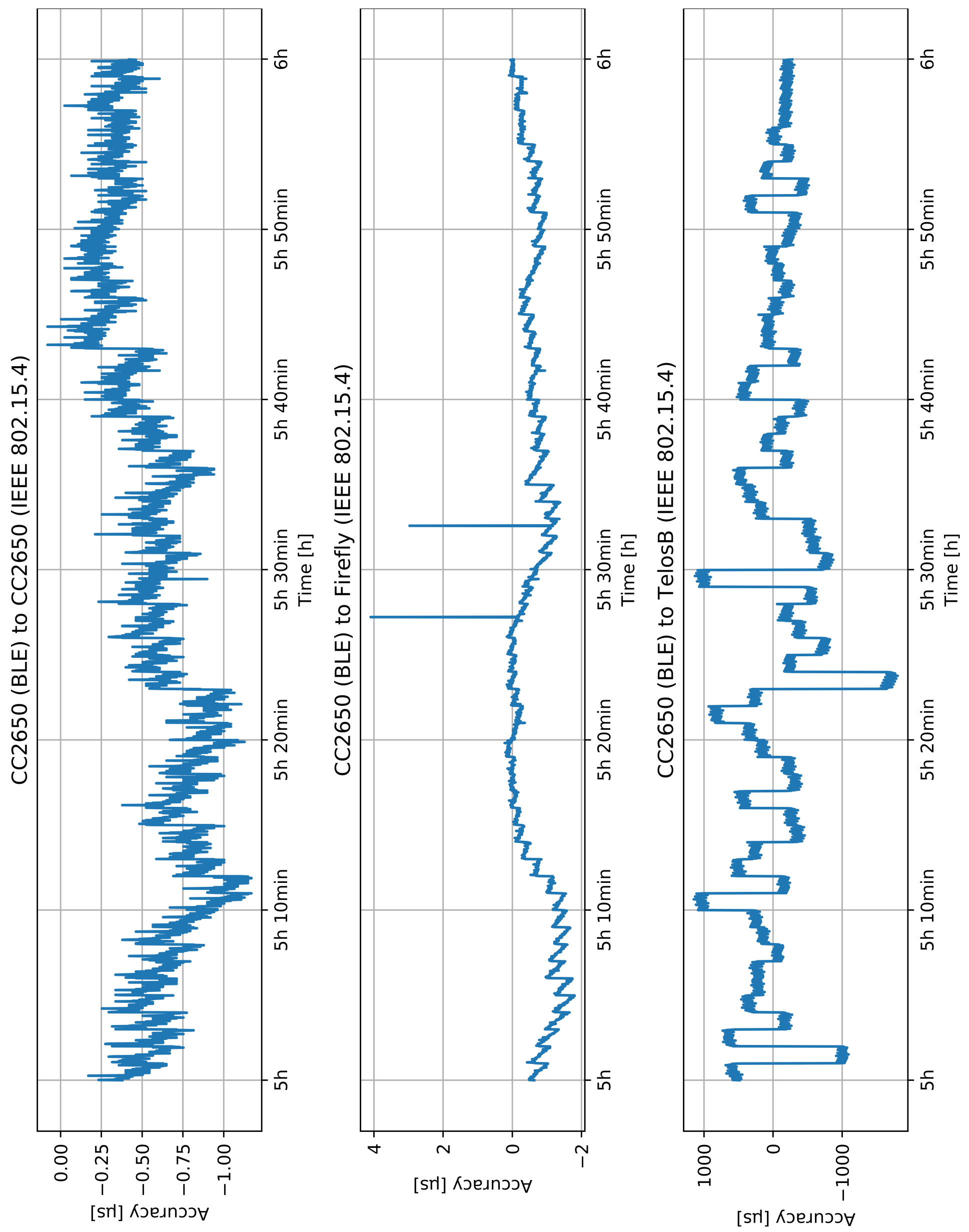
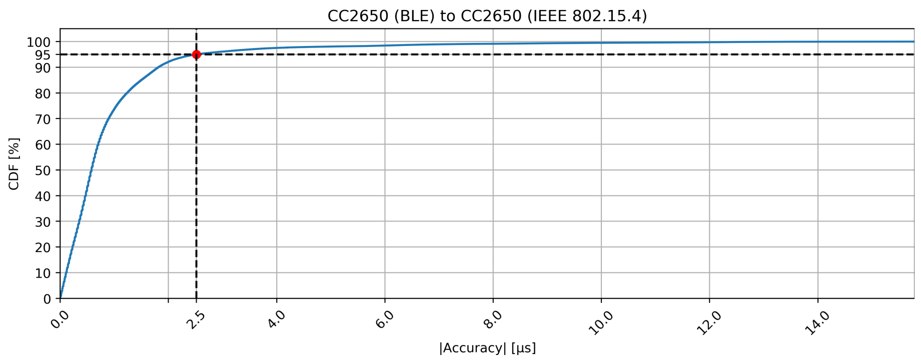
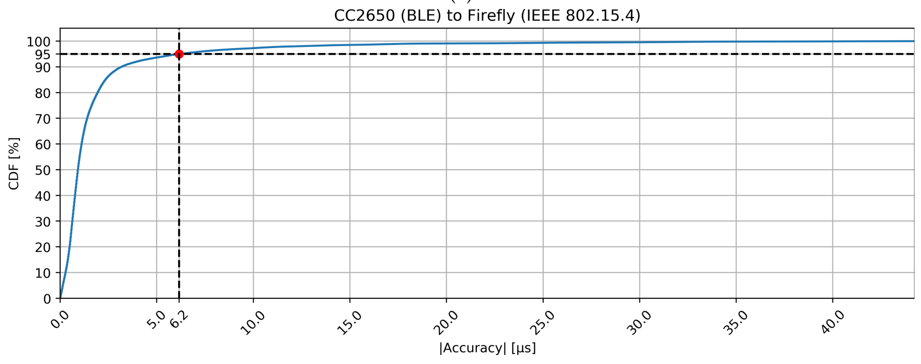


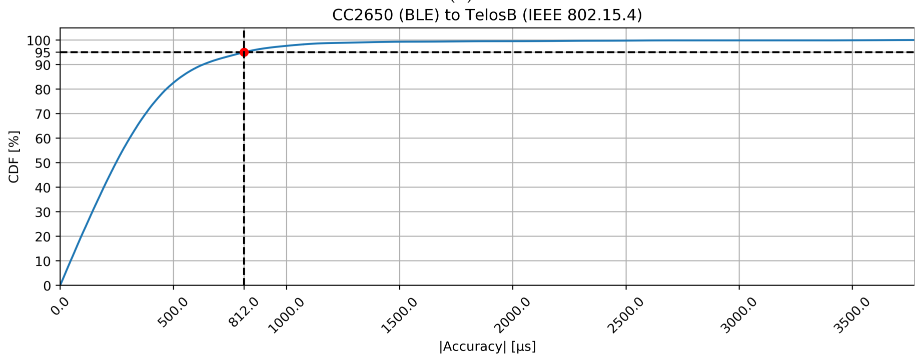
Figure 5.9: Time-domain plot of the long-term X-Sync evaluation zoomed between t=5h and t=6h. Hereby the CC2650 node in BLE mode was used as a transmitter and all other nodes were configured as receivers, forming a broadcast configuration.



(a)



(b)



(c)

Figure 5.10: CDF of the modulus of the relative synchronization error of the long-term X-Sync evaluation seen in Figure 5.8. Hereby the CC2650 node in BLE mode was used as a transmitter and all other nodes were configured as receivers, forming a broadcast configuration.

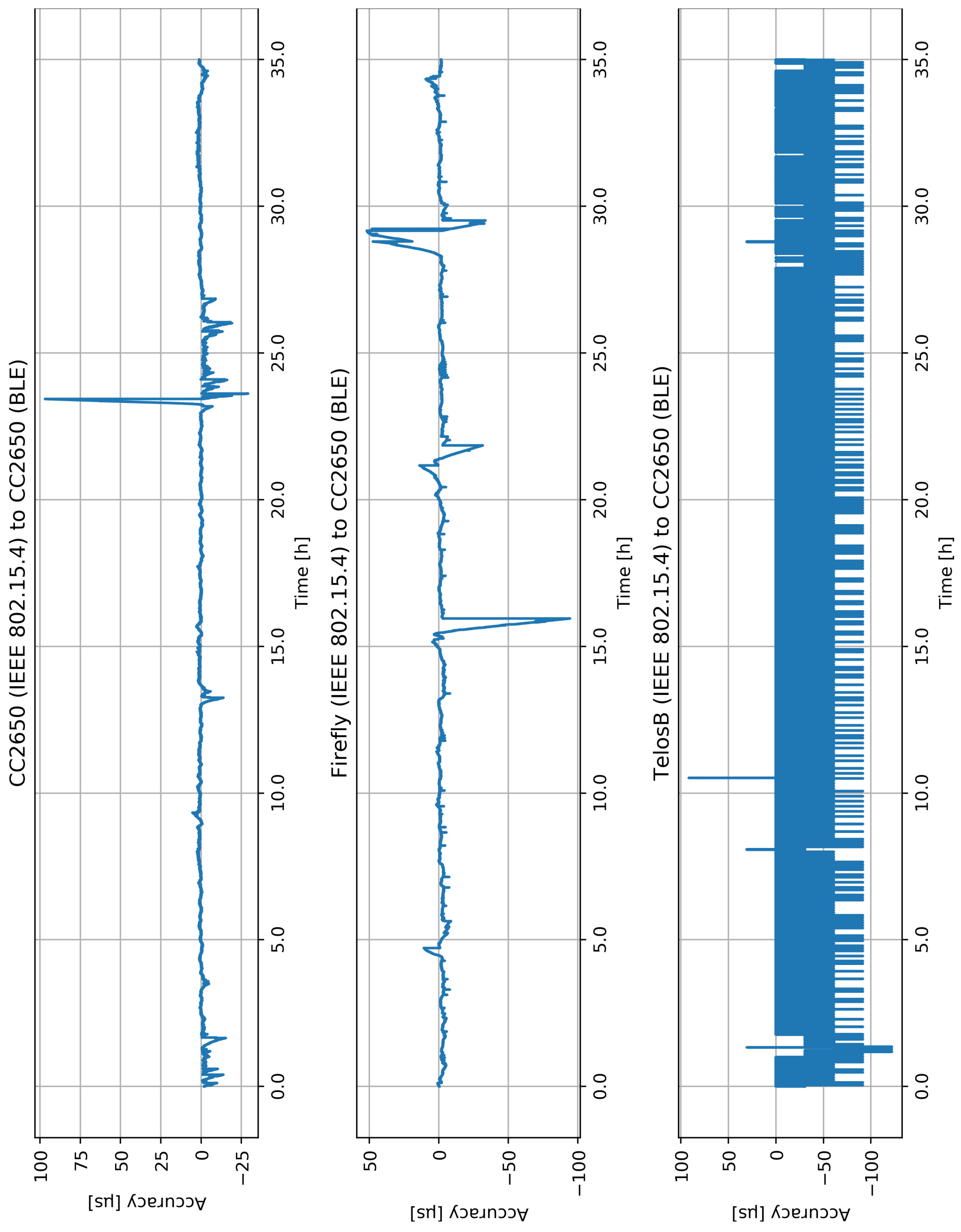


Figure 5.11: Time-domain plot of the long-term X-Sync evaluation. Hereby the CC2650 node in BLE mode was configured as receiver and all other nodes were configured one-by-one as transmitters, forming a unicast configuration.

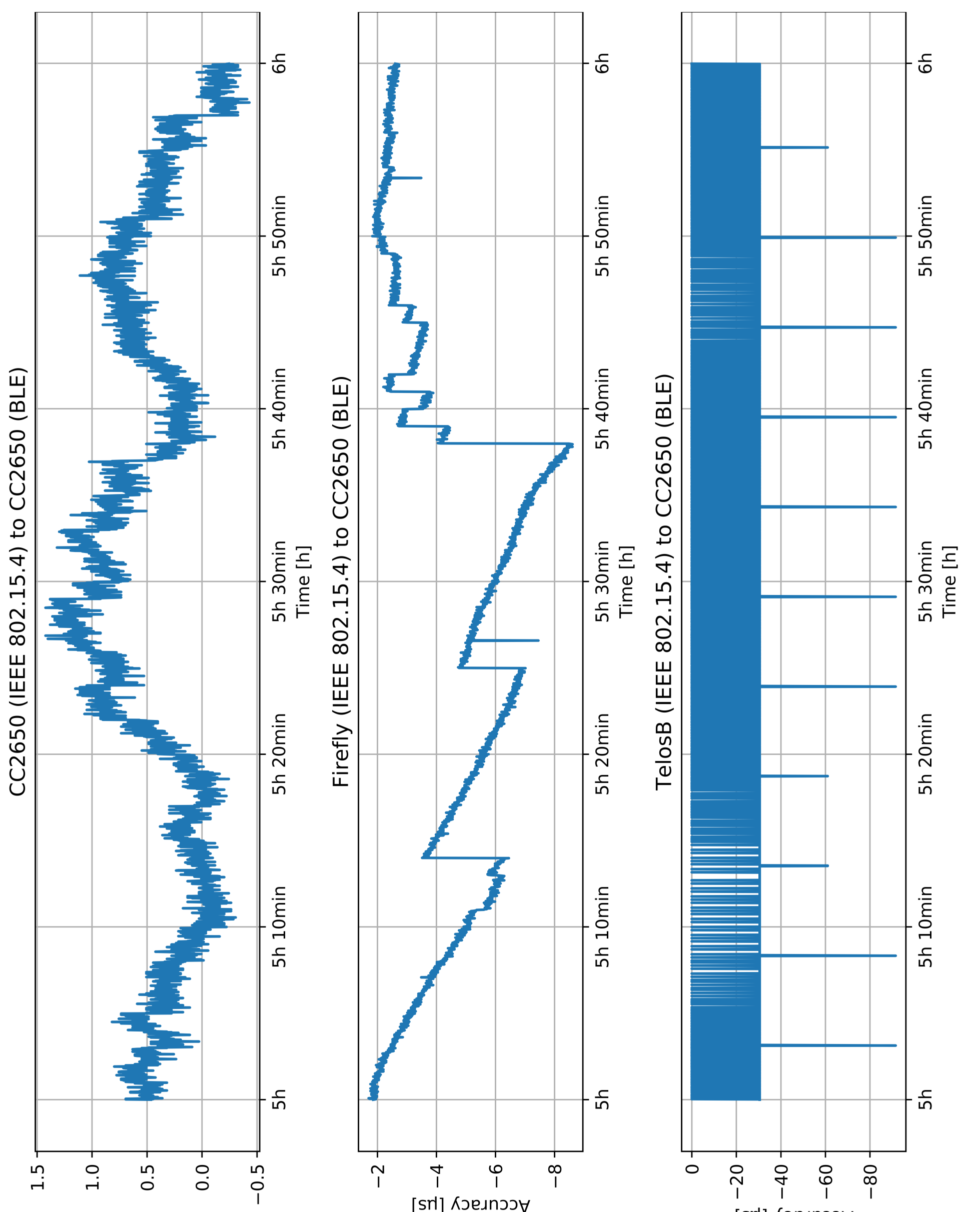
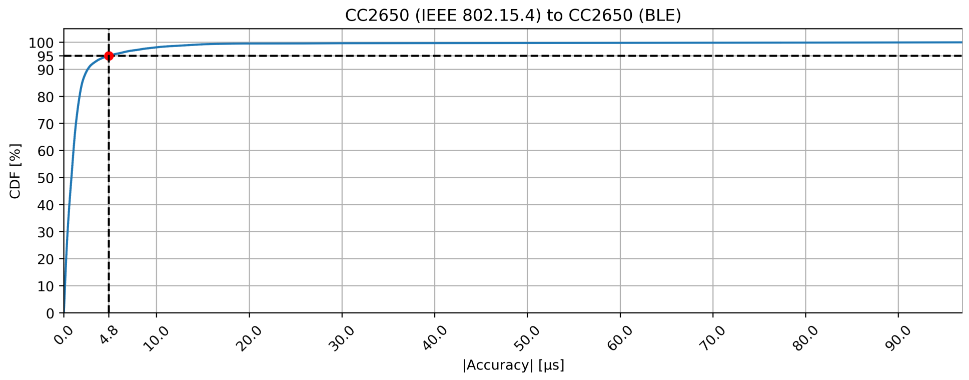
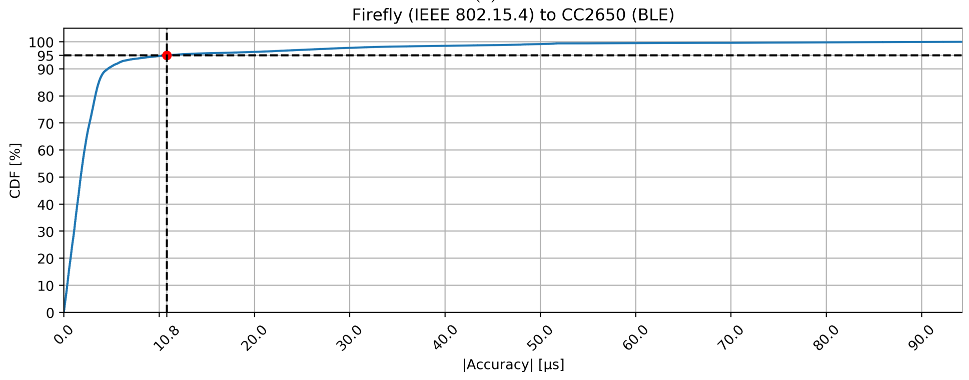


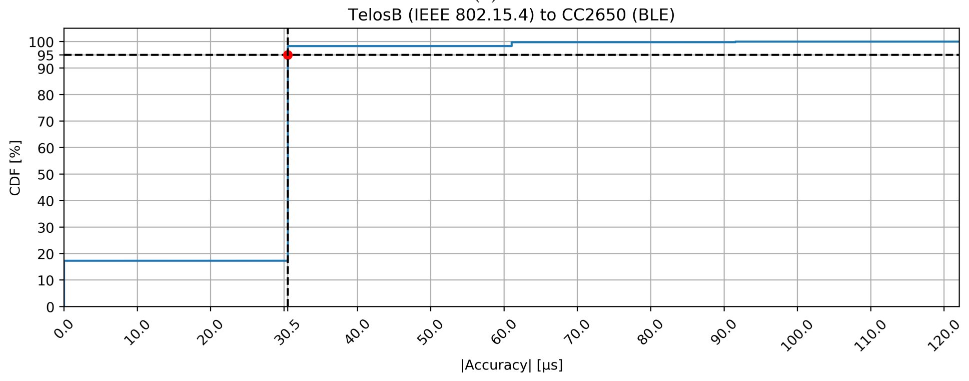
Figure 5.12: Time-domain plot of the long-term X-Sync evaluation zoomed between t=5h and t=6h. Hereby the CC2650 node in BLE mode was configured as receiver and all other nodes were configured one-by-one as transmitters, forming a unicast configuration.



(a)



(b)



(c)

Figure 5.13: CDF of the modulus of the relative synchronization error of the long-term X-Sync evaluation seen in Figure 5.11. Hereby the CC2650 node in BLE mode was configured as receiver and all other nodes were configured one-by-one as transmitters, forming a unicast configuration.

6 Conclusion and Future Work

This chapter concludes the thesis with a summary of the contributions of this thesis in Section 6.1 and an outlook about the future development of X-Sync in Section 6.2.

6.1 Conclusion

In this thesis, we present X-Sync, a novel and accurate cross-technology clock synchronization scheme between off-the-shelf BLE and IEEE 802.15.4 devices. Compared to other works in the field, X-Sync offers a superior synchronization accuracy, of under $1\mu s$ (CC2650 and Firefly), depending on the configuration. It also supports low-computing power nodes, such as the TelosB, where a synchronization accuracy below $100\mu s$ can be achieved. These results are achieved by adapting the packet-based CTC scheme X-Burst [42] for time critical applications. Thereby, we analysed the delays present in packet-based CTCS and then combined MAC timestamping mechanisms with novel CTC delay compensation schemes in order to compensate and correct these delays. Furthermore, we used sophisticated filtering and clock skew estimation algorithms to further improve the relative synchronization accuracy.
X-Sync was integrated into the open source operating system Contiki-NG, by adding and altering blocks from the modular architecture of X-Burst. We hereby change the message format and introduce a synchronization preamble, which is used to compensate the sampling delay inherent in packet-based CTCS systems. Throughout all blocks, a special focus on accurate timing was taken.
X-Sync was evaluated on real hardware showing a working cross-technology synchronization between a ZigBee and a BLE device. The evaluation showed that depending on the configuration, the relative synchronization error can be lower than $1\mu s$ on the CC2650 and Firefly platform and lower than $100\mu s$ on the TelosB platform. Furthermore, the memory footprint, the energy consumption and the reliability of X-Sync were evaluated. Lastly, a long-term evaluation showed that X-Sync is able to maintain the specified

accuracies also in real-world applications and that X-Sync is able to cope with disturbances on the shared CTC channel.

6.2 Future Work

In the following, an outlook about the future development of X-Sync is given.

Implement a high-level time synchronization scheme on top. While the evaluation shows a very good synchronization accuracy, it also shows that the average synchronization error is sometimes significant, depending on the configuration. The reasons for this effect need to be further analysed and corrected. One possible explanation could be that a non-constant transmission power is used, or the receivers automatic gain control is active. Regardless of the reason, it results in non-static environmental conditions. By using a simple round-trip synchronization algorithm on top of X-Sync we could drastically improve this situation. Even more sophisticated algorithm could be ported in order to achieve an even better synchronization accuracy.

Interoperability with X-Burst. During the implementation of X-Sync, the X-Burst message format was changed in order to include the synchronization preamble as well as some additional fields. This effectively means that X-Sync is not compatible with X-Burst anymore. By reordering the fields, and sending the synchronization preamble at a later point in time, interoperability would be possible.

Adding Wi-Fi support. X-Burst was already also ported to Wi-Fi [22] and therefore, the next logical step is to also port X-Sync to Wi-Fi. One mayor challenge hereby is the time-critical HAL layer, which also has to be implemented.

Port X-Sync to another hardware platform. So far, X-Sync is implemented on three hardware platforms with only one supporting BLE. The next logical step is to port X-Sync to another BLE node.

Software-defined radio implementation. In order to simulate, observe and debug ongoing transmissions, a software defined radio implementation of X-Sync would be very helpful. It could help in improving the timing and thereby the relative synchronization accuracy.

Bibliography

[1] H. Hejazi *et al.*, "Survey of platforms for massive IoT," in *IEEE International Conference on Future IoT Technologies (Future IoT)*, 2018.

[2] P. Scully, *The Top 10 IoT Segments in 2018 - based on 1,600 real IoT Projects*, 2019. [Online]. Available: `https://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/` (visited on 09/30/2018).

[3] T. Alam, "A reliable communication framework and its use in Internet of Things (IoT)," 2018.

[4] F. Gao *et al.*, "Design and optimization of a cross-layer routing protocol for multi-hop wireless sensor networks," in *Proc. of the International Conference on Sensor Network Security Technology and Privacy Communication System*, 2013.

[5] Q. M. Chaudhari *et al.*, "Estimation of Clock Parameters for Synchronization in Wireless Sensor Networks," in *Proc. of the IEEE International Conference on Signal Processing and Communications*, 2007.

[6] M. Buettner *et al.*, "X-mac: A short preamble mac protocol for duty-cycled wireless sensor networks," in *Proc. of the 4th International Conference on Embedded Networked Sensor Systems*, 2006.

[7] J. D. Case *et al.*, *Simple Network Management Protocol (SNMP)*, 1990.

[8] D. P. Shepard *et al.*, "Evaluation of the vulnerability of phasor measurement units to GPS spoofing attacks," *International Journal of Critical Infrastructure Protection*, 2012.

[9] R. Tan *et al.*, "Impact of Integrity Attacks on Real-time Pricing in Smart Grids," in *Proc. of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, 2013.

[10] X. Guo *et al.*, "ZIGFI: Harnessing Channel State Information for Cross-Technology Communication," *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018.

[11] G. Werner-Allen *et al.*, "Monitoring Volcanic Eruptions with a Wireless Sensor Network," in *Proc. of the Second European Workshop on Wireless Sensor Networks*, 2005.

[12] S. Kim *et al.*, “Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks,” in *Proc. of the Sixth International Symposium on Information Processing in Sensor Networks*, 2007.

[13] K. Khanchuea and R. Siripokarpirom, “A Multi-Protocol IoT Gateway and WiFi/BLE Sensor Nodes for Smart Home and Building Automation: Design and Implementation,” in *Proc. of the 10th International Conference of Information and Communication Technology for Embedded Systems*, 2019.

[14] M. Hawelikar and S. Tamhankar, “A design of Linux based ZigBee and Bluetooth low energy wireless gateway for remote parameter monitoring,” in *Proc. of the 2015 International Conference on Circuits, Power and Computing Technologies*, 2015.

[15] G. Aloi *et al.*, “A Mobile Multi-Technology Gateway to Enable IoT Interoperability,” in *Proc. of the First International Conference on Internet-of-Things Design and Implementation*, 2016.

[16] T. Zachariah *et al.*, “The Internet of Things Has a Gateway Problem,” 2015.

[17] K. Chebrolu *et al.*, “Esense: Communication through energy sensing,” in *Proc. of the 15th annual international conference on Mobile computing and networking*, 2009.

[18] X. Zhang and K. G. Shin, “Gap Sense: Lightweight coordination of heterogeneous wireless devices,” in *Proc. of the IEEE International Conference on Computer Communications*, 2013.

[19] ——, “Cooperative Carrier Signaling: Harmonizing Coexisting WPAN and WLAN Devices,” *IEEE/ACM Transactions on Networking*, 2013.

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

[21] W. Jiang *et al.*, “BlueBee: A 10,000x Faster Cross-Technology Communication via PHY Emulation,” in *Proc. of the 15th ACM Conference on Embedded Network Sensor Systems*, 2017.

[22] H. Brunner *et al.*, "Cross-Technology Broadcast Communication between Off-The-Shelf Wi-Fi, BLE, and IEEE 802.15.4 Devices," in *Proc. of the 17th International Conference on Embedded Wireless Systems and Networks*, 2020.

[23] *CC256x Dual-Mode Bluetooth Controller*, SWRS121E, Texas Instruments, 2012.

[24] *Tmote Sky Low Power Wireless Sensor Module*, Moteiv Corporation, 2006.

[25] *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*, ZOL-BO001-A2, Zolertia, 2017.

[26] F. Mattern and C. Floerkemeier, "From Active Data Management to Event-based Systems and More," in, 2010, ch. From the Internet of Computers to the Internet of Things.

[27] D. Tomtsis *et al.*, "Evaluating existing wireless technologies for IoT data transferring," in *Proc. of the South Eastern European Design Automation, Computer Engineering, Computer Networks and Social Media Conference*, 2017.

[28] J. d. C. Silva *et al.*, "LoRaWAN - A low power WAN protocol for Internet of Things: A review and opportunities," in *Proc. of the 2nd International Multidisciplinary Conference on Computer and Energy Science*, 2017.

[29] "IEEE Standard for Low-Rate Wireless Networks," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, 2016.

[30] *ZigBee Specification*, ZigBee Standards Organization, 2012. [Online]. Available: `http://www.zigbee.org/wp-content/uploads/2014/11/docs-05-3474-20-0csg-zigbee-specification.pdf` (visited on 03/03/2019).

[31] P. Zand *et al.*, "ISA100.11a: The ISA100.11a extension for supporting energy-harvested I/O devices," in *Proc. of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2014.

[32] J. Song *et al.*, "WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control," in *Proc. of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, 2008.

[33] Z. Shelby and C. Bormann, *6LoWPAN: The Wireless Embedded Internet*. Wiley Publishing, 2010.

[34] D. L. Mills, *Computer Network Time Synchronization: The Network Time Protocol*. CRC Press, Inc., 2006.

[35] J. Elson *et al.*, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review*, 2002.

[36] C. Gu *et al.*, "Broadcast time synchronization algorithm for wireless sensor networks," 2006.

[37] S. Ganeriwal *et al.*, "Timing-sync Protocol for Sensor Networks," in *Proc. of the 1st International Conference on Embedded Networked Sensor Systems*, 2003.

[38] M. Maróti *et al.*, "The Flooding Time Synchronization Protocol," in *Proc. of the 2nd international conference on Embedded networked sensor systems*, 2004.

[39] S. M. Kim *et al.*, "FreeBee: Cross-technology Communication via Free Side-channel," in *Proc. of the 21st Annual International Conference on Mobile Computing and Networking*, 2015.

[40] *CC13x0, CC26x0 SimpleLink Wireless MCU*, SWCU117H, Texas Instruments, 2015.

[41] Z. Li and T. He, "Webee: Physical-layer cross-technology communication via emulation," in *Proc. of the 23rd Annual International Conference on Mobile Computing and Networking*, 2017.

[42] R. Hofmann *et al.*, "X-Burst: Enabling Multi-Platform Cross-Technology Communication between Constrained IoT Devices," in *Proc. of the 16th Annual IEEE International Conference on Sensing, Communication, and Networking*, 2019.

[43] J. Bauwens *et al.*, "Coexistence between IEEE802.15.4 and IEEE802.11 through cross-technology signaling," in *Proc. of the IEEE Conference on Computer Communications Workshops*, 2017.

[44] ——, "Demo abstract: Cross-technology TDMA synchronization using energy pattern beacons," in *Proc. of the IEEE Conference on Computer Communications Workshops*, 2017.

[45] P. Ruckebusch *et al.*, "Cross-technology wireless experimentation: Improving 802.11 and 802.15.4e coexistence," in *Proc. of the IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2016.

[46] P. Valck *et al.*, "Exploiting programmable architectures for wifi/zigbee inter-technology cooperation," *Eurasip Journal on Wireless Communications and Networking*, 2014.

[47] I. Tinnirello *et al.*, "Wireless mac processors: Programming mac protocols on commodity hardware," in *Proc. of the IEEE International Conference on Computer Communications*, 2012.

[48] P. d. Mil *et al.*, "Snapmac: A generic mac/phy architecture enabling flexible mac design," *Ad Hoc Networks*, 2014.

[49] Z. Yu *et al.*, "Crocs: Cross-Technology Clock Synchronization for WiFi and ZigBee," in *Proc. of the International Conference on Embedded Wireless Systems and Networks*, 2018.

[50] *TelosB Mote Platform*, 6020-0094-03, MEMSIC.

[51] *2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*, SWRS041c, Texas Instruments, 2014.

[52] Advanticsys, *Advanticsys MTM-CM5000-MSP*. [Online]. Available: `https://www.advanticsys.com/shop/mtmcm5000msp-p-14.html` (visited on 09/30/2018).

[53] *CC2538 Powerful Wireless Microcontroller System-On-Chip for 2.4 GHz IEEE 802.15.4, 6LoWPAN, and ZigBee Applications*, SWRS096D, Texas Instruments, 2012.

[54] *CC1200 Low-Power, High Performance RF Transceiver*, SWRS123D, Texas Instruments, 2013.

[55] *Contiki-NG GIT Repository*. [Online]. Available: `https://github.com/contiki-ng/contiki-ng` (visited on 09/30/2018).

[56] A. Dunkels *et al.*, "Protothreads: Simplifying event-driven programming of memory-constrained embedded systems," in *Proc. of the 4th International Conference on Embedded Networked Sensor Systems*.

[57] S. M. Lasassmeh and J. M. Conrad, "Time synchronization in wireless sensor networks: A survey," in *Proc. of the IEEE SoutheastCon*, 2010.

[58] D. Petrov *et al.*, "Distributed GNSS-based Time Synchronization and applications," in *Proc. of the 8th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops*, 2016.

[59] A. Flockett, *Time-sensitive networking: reliable communication for Industrial IoT*, 2017. [Online]. Available: `https://www.electronicspecifier.com/blog/time-sensitive-networking-reliable-communication-for-industrial-iot` (visited on 09/30/2018).

[60] A. Mahmood *et al.*, "Methods and performance aspects for wireless clock synchronization in IEEE 802.11 for the IoT," in *Proc. of the IEEE World Conference on Factory Communication Systems*, 2016.

[61] M. A. Fischler *et al.*, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, 1981.

[62] H. Wang *et al.*, "Maximum likelihood estimation of clock skew in wireless sensor networks with periodical clock correction under exponential delays," *IEEE Transactions on Signal Processing*, 2017.