Master's Thesis

Time Synchronization in Short Range Wireless Networks

Filip Gummesson
Kristoffer Hilmersson

Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, 2016.

Time Synchronization in Short Range Wireless Networks

Filip Gummesson
Kristoffer Hilmersson

u-blox
Östra Varvsgatan 4
211 75 Malmö

Advisors:
Mats Cedervall, EIT
Mats Andersson, u-blox

June 23, 2016

Printed in Sweden
E-huset, Lund, 2016

Abstract

Energy efficient wireless devices is a trend that has been on the rise the last few years. Energy efficiency properties may fill a function in wireless sensor networks where the devices could run on coin-cell batteries and still last for years. To make sense of the sensor data collected, there is a requirement of accurate time synchronization in the network. This report focuses on investigating if present time synchronization standards, NTP and PTP, and de facto-standards, RBS, TPSN, and FTSP, can provide sufficient accuracy in a network consisting of devices where energy efficiency and mobility are the ground pillars, or if accurate time synchronization and the constraints of highly energy efficient devices proves contradictory.

The knowledge gained from studying the standards lay the foundation of a case study where RBS was implemented in a network consisting of Bluetooth low energy devices, proving that low millisecond accurate time synchronization was achievable using application based timestamping. Compared to wireless sensor networks consisting of devices not developed with today's high energy efficiency focus, the accuracy is a factor thousand less precise, but with some hardware features, like hardware based timestamping and a more accurate clock, it would not be impossible to reduce the gap.

Acknowledgement

We would like to express our deepest thanks to u-blox and all their employees in Malmö, especially our supervisor Mats Andersson, who have shared his precious time with us. They have assisted us with technical detail, guidance, and valuable input on our project.

We would also like to express our sincere thanks and appreciation to our supervisor Mats Cedervall at the Faculty of Engineering, Lunds University, who has guided us through the project with helpful advice, but also asked those tough questions that made us think one extra time.

Table of Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
BLE	Bluetooth Low Energy
COTS	Commercial-Off-The-Shelf
FTSP	Flooding Time Synchronization Protocol
GATT	Generic Attribute Profile
HCI	Host Controller Interface
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
MAC	Media Access Control
Mbps	Mbit/s
NTP	Network Time Protocol
PHY	Physical Layer
PLCP	Physical Layer Convergence Procedure
PMD	Physical Medium Dependent
PTP	Precision Time Protocol
RBS	Reference Broadcast Synchronization
RTC	Real-time clock
SDIO	Secure Digital Input Output
TPSN	Timing-sync Protocol for Sensor Networks
UART	Universal Asynchronous Receiver/Transmitter
UUID	Universally Unique Identifier
WSN	Wireless Sensor Network

Chapter 1

Introduction

The revolution of the Internet of Things (IoT) is upon us. Everyday objects, like toasters and coffee makers, suddenly have the need to be controlled by one's smartphone or computer. Even though this might be a "must have feature", one still has to get out of bed in the morning to put a cup under the coffee maker, and a toast in the toaster.

Of course, there are IoT applications that really matters too. A Wireless network consisting of small sensors is such an application. These networks are called Wireless Sensor Networks (WSN). A sensor, or node, in a WSN, is an embedded system probably communicating with either:

- WiFi
- Bluetooth
- Bluetooth low energy

Embedded systems are highly related to the IoT revolution; hardware is cheaper than ever, and these communication protocols are so energy efficient that they could run on a single button cell battery for years.

Yesterdays sensors needed cable connections to a central unit where data, where collected and, analysed. Today, deploying a sensor is as easy as mounting the sensor in place and pressing a button. The sensor measurements are transmitted using WiFi, Bluetooth, or Bluetooth low energy (BLE) over a mesh network to the central unit. But, there are problems with wireless communication. To make the measurement meaningful, a sensor needs to take a timestamp together with each measurement that is transmitted to the central. In this way, measurements can be arranged in order of occurrence.

All embedded systems have real-time clocks that are used in the process of timestamping, but the question is how much the clock differs from the other nodes in the WSN. There has to be a common timescale within the WSN.

1.1 Background

Standards for time synchronization, like IEEE 1588, Precision Time Protocol (PTP), have most often been implemented using wired networks, such as IEEE 802.3 (Ethernet). But technology advancements in recent years have shown that precise time synchronization could also be achieved through wireless communication using WiFi, Bluetooth, or BLE. If time synchronization using wireless data links could reach acceptable precision, it would remove the wiring installation costs, and result in more flexible systems, due to the cut out of physical limitation that cables entail.

This project is carried out in cooperation with u-blox, a Swiss company that creates wireless semiconductors and modules to connect and locate people, vehicles, and machines. At the u-blox office in Malmö, the research, and development are focused on short-range connectivity like WiFi and Bluetooth low energy.

1.2 Objectives

The objectives for this thesis is to:

- Investigate if there are any standards or de facto standard methods that can be used, and evaluate the accuracy that can be achieved in a short-range wireless network over:
  - WiFi
  - Bluetooth
  - Bluetooth low energy
- Propose solutions for achieving as precise time synchronization as possible on energy constrained embedded devices.
- Conduct a case study on u-blox products and measure performance.

1.3 Previous Work

Lamport's paper on logical clocks marked the beginning of time synchronization, or rather, ordering of events occurring in a distributed system [1]. Since then, it has been a topic of discussion on how to synchronize clocks in distributed systems. Multiple protocols for time synchronization have been developed and implemented [2–4].

Back in 1987, the authors of [2] presented the first synchronization algorithm where logical clocks had the same accuracy as its underlying physical clock.

Mill's network time protocol (NTP) [3] has kept the clocks of the internet in sync since 1984. NTP distinguishes itself from other protocols due to its scalability, self-configuration in large networks with hierarchical structures, and its robustness to failure.

For computer networks connected with Ethernet cables, the Precise Time Protocol (PTP) has been developed [4]. It is designed to give higher accuracy than NTP in a local area network.

We are in a new era where networks consist of small systems with constraints on processing power and energy consumption. The synchronization protocols need to be light weight but still manage to synchronize WSNs. In [5], the authors argue that a method like NTP would be ill-suited for this kind of networks. They instead list design principles for such a time synchronization algorithm. They state that it should be possible to have tunable modes, since unnecessary precision waste resources; no global timescale should be present in the network, instead each node should have a table to convert from its clock to any other node's time in a network; let nodes run unsynchronized, and let the nodes in a WSN agree on a time after an event has occurred; let the application determine the degree of synchronization, because some applications do not need as high accuracy as others.

1.4 Delimitations

This thesis will cover the standards and de facto standards available, they will be evaluated and analyzed on performance. It also includes a detailed description of the case study that has been part of this project.

The thesis will not directly focus on how to modify hardware, nor lower software layers, to reach higher precision in terms of time synchronization. The security aspects of time synchronization algorithms are not studied. Neither has the power consumption been optimized during the implementation phase.

1.5 Thesis Outline

The rest of this report is organized as follows: Chapter 2 presents the time synchronization problem and parts of the communication protocols that are relevant to the problem with the main focus on BLE. The theory concerning Network Time Protocol, Precision Time Protocol, Reference Broadcast Synchronization, Timing-Sync Protocol for Sensor Networks, Flooding Time Synchronization Protocol, and Broadcast Synchronization over Bluetooth will be explained, these are the time synchronization standards and de facto-standards that this report is focused on.

Chapter 3 contains the methodology, where an approach on implementing a time synchronization algorithm on u-blox's modules is presented. Chapter 4 compiles relevant data derived from the literature study and shows the result of the case study.

Chapter 5 contains a discussion and conclusions in regard to the results from Chapter 4.

Chapter 2

Theory

In this chapter, the theory behind time synchronization in wireless sensor networks is presented. The first part gives some general information about the synchronization problem. The second part presents the communication protocols (WiFi, Bluetooth, and Bluetooth low energy) and how to use them to synchronize nodes in a WSN. Lastly, standards and de facto standards are described.

2.1 What is an Embedded System?

An embedded system is a microcontroller based system which is designed to carry out one or a few tasks. Embedded systems can be found in all kinds of electrical devices, e.g., cars or other vehicle, washing machines, and TVs. The list is very long, actually, 98% of all manufactured microprocessors are components of embedded systems [6].

Embedded systems can be characterized by their small sizes, low cost, and low power consumption compared to a general purpose computer. However, these characteristics put constraints on embedded systems in terms of processing power and memory size.

2.1.1 Wireless Sensor Network

A Wireless Sensor Network (WSN) is, in the context of this report, a group of embedded systems with a communication infrastructure for monitoring and recording conditions. The embedded systems in a WSN are called nodes. It is usually preferred that the sensor nodes are small and portable. Potential implementations of WSNs are:

- Industrial automation
- Home automation
- Medical device monitoring
- Temperature monitoring
- Body sensors
- Robot control

2.2 Time synchronization

All computer clocks are dependent on a crystal oscillator, a clock in an embedded system is no different. A crystal oscillator is an electronic circuit that uses the mechanical resonance of a vibrating crystal to create an electrical signal with a precise frequency. The crystal consists of piezoelectric material. The oscillator assists the embedded system's Real Time Clock (RTC) to tick. Tick, after tick.

The RTC is an approximation of real time, t, and can be written as $C(t)$ [7]. $C(t)$ can be subject to both phase error and clock skew. The phase offset is easily thought of as the difference in between UTC (Coordinated Universal Time) and what one's wrist watch displays. The hardware oscillator is the source of error when it comes to clock skew. If the oscillator's frequency is too high, or too low, the offset from our approximation $C(t)$ to UTC will increase by time.

2.2.1 Three Types of Synchronization

Time synchronization algorithms can be categorized into three different types [7]. First, and most simple, is to be able to order events E_1 and E_2 on which occurred before the other. This can be done by comparing local clocks, instead of having them synchronized. The second category is the ability to keep their relative clock speed. Here, each node in the network runs its local clock freely but keeps information about all other clocks to, at any time, convert one clock to another. Third, and most complex is to make all nodes in a network have the same perception of time. Most time synchronization algorithms belong to the second domain, e.g., the algorithm implemented during the case study (see Chapter 3).

2.2.2 Timestamping

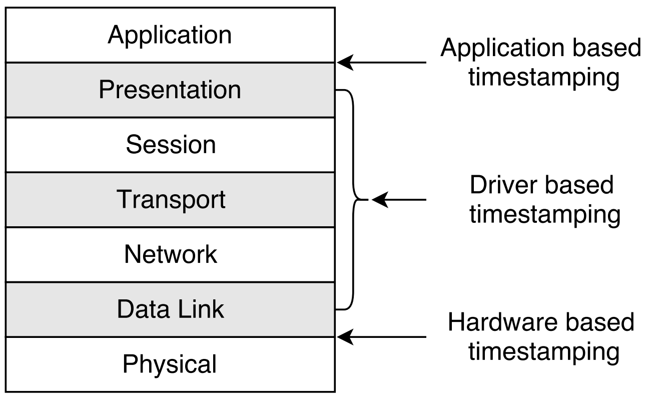
Application
Presentation
Session
Transport
Network
Data Link
Physical
Application based timestamping
Driver based timestamping
Hardware based timestamping

Figure 2.1: Possible location for timestamping in reference to the OSI-model.

A timestamp contains information about when a certain event occurred. The information can range from giving a date to being as accurate as a small fraction of a second. Timestamping is an essential part of time synchronization of networks. In terms of effectively timestamping network events, it is a question of where or how early in the process you can timestamp to minimize the non-deterministic jitter presented during data propagation from the physical layer to the application layer when using the OSI-model as reference (see Figure 2.1). In [8], the authors have decomposed the process of transmitting and receiving a data packet from one node to another into six phases. The phases are listed below and illustrated in Figure 2.2. The figure is to give an idea of how the traversing time is related to each phase, it does not, however, correspond to their actual ratio.

- *Send Time*

  When a node, in the network, decides to transmit a packet. The packet is scheduled to be sent. In this phase, time is spent on constructing the actual packet in the Application layer (analogy to the OSI model). The packet is propagated down the stack to be transmitted. Each layer adds to the nondeterminism.

- *Access Time*

  In the MAC layer, the packet waits for the radio link to become available. The radio link can be unavailable if the radio is receiving data or there are other packets to be transmitted. The time a packet waits for the radio link may vary, hence add to the nondeterminism.

- *Transmission Time*

  The transmission time involves the time it takes to transmit a packet, bit by bit, in the physical layer. This process is non-deterministic, but will not affect the time synchronization as much as other phases.

- *Propagation Time*

  This is the time it takes a radio signal to propagate through air. Since electromagnetic waves propagate through the air with the speed of light, this phase is highly deterministic. The latency is imperceptible compared to other phases, hence it can be neglected.

- *Reception Time*

  The time it takes to receive bits with the radio receiver. Just like the transmission time, this time, is deterministic in most cases.

- *Receive Time*

  The received bits are concatenated to a packet in the lower stack layers. The receive time is the time of constructing a packet from bits, and the time it takes for a packet to arrive at the Application layer. The propagation to the highest level is non-deterministic.

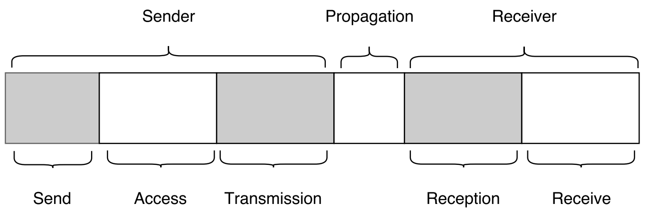

Figure 2.2: Decomposition of packet delay over a wireless link.

Timestamping as close to the physical layer as possible would mean that the only difference between two nodes would be the propagation time. This is always preferred since it minimizes the nondeterministic behavior. However, this might not be possible depending on what device one is using since hardware based timestamping requires support in hardware. Driver based timestamping is preferably implemented as close to the physical layer as possible, for every layer ranging from application to physical non-deterministic jitter can be reduced. Performing application based timestamping is the easiest option but also most uncertain. For some use cases, it provides efficient enough results.

2.3 A Perfect Clock

In [7], the rate of a perfect RTC is described as $\frac{dC}{dt}$, which should equal 1. Due to physical effects to the hardware oscillator, that is however not possible. The pace of the oscillator is not fixed; hence does the real time clock skew. Another approximation of $C(t)$ is needed. We write $C(t)$ for any node i that is in the network, as

$$C_i(t) = a_i t + b_i \tag{2.1}$$

where $a_i(t)$ and $b_i(t)$ denotes clock skew and phase offset respectively.

2.4 802.11 WLAN

IEEE 802.11 is a collection of standards for WLAN, also known as Wireless Ethernet. The standards are created and maintained by the IEEE. To this date, 16 versions of 802.11 have been released. 802.11b, 802.11g, 802.11n, and 802.11ac are the versions that have had the biggest impact, transmitting data at a speed of 11, 54, 270, and 433 Mbps respectively. The most recent and widely used is 802.11ac. In this section, the focus will be on the essential parts, related to efficient time synchronization, of the 802.11 protocol.

2.4.1 Physical Layer

The physical layer (PHY) is the first and lowest layer and responsible for putting bits "on the air". The PHY is divided into two sublayers: the Physical Layer

Convergence Procedure (PLCP) sublayer and the Physical Medium Dependent (PMD) sublayer [9]. PLCP adds its own header to frames from the MAC and can be seen as the link between the MAC layer and the radio transmission in the air. The PMD's role is to transmit any bits it receives from the PLCP into the air using the antenna. The partition of the Physical layer in terms of PLCP and PMD is illustrated in Figure 2.3.

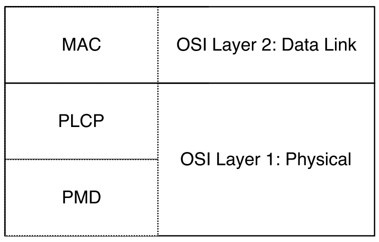

Figure 2.3: Physical layer architecture with sublayers.

2.4.2 Media Access Control

The media access control (MAC) layer is the lowest sublayer of the data link layer (Figure 2.4). The MAC can be seen as the interface between the logical link layer (LLC) and the PHY. The MAC's major responsibility is to offer reliable link-to-link data transfer.

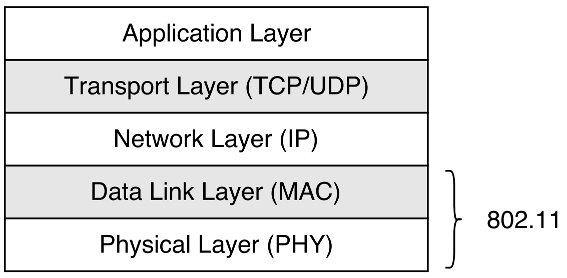

Figure 2.4: 802.11's relation to the OSI model. The layers above the Data Link layer is not affected by the standard.

2.5 Bluetooth

The latest version of Bluetooth goes under the name of Bluetooth v4.2 [10] and was released on December 2, 2014. v4 includes Classic Bluetooth, Bluetooth high speed, and Bluetooth low energy (BLE) protocols. The major focus of this report is Bluetooth Low Energy.

2.5.1 Comparison

The Bluetooth specification defines three different protocols, Bluetooth high speed has not been taken into account and classic Bluetooth is only used for comparison. Some specifications is compiled for comparison purposes between classic Bluetooth and BLE in Table 2.1 derived from [10].

Table 2.1: Bluetooth 4.2, A comparison of technical specification

Technical Specification	Classic Bluetooth	BLE
Over the air data rate	1 - 3 Mbits	1 Mbits
Application throughput	0.7 - 2.1 Mbits	0.27 Mbits
Network topology	Scatternet	Scatternet
Power consumption	1 W as reference	0.01 to 0.5 W
Peak current consumption	<30 mA	<15 mA

2.6 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a wireless personal network technology designed as both a complementary technology to classic Bluetooth as well as the lowest possible power wireless technology that can be designed and built with today's technology [11]. The wireless market has always focused on increasing data rates and so is also the case with Classic Bluetooth. BLE takes a completely different direction, instead focusing on ultra-low power consumption. Thus making it a very interesting candidate for the laptop, mobile, and IoT industry that strive for longer lasting battery life.

2.6.1 Architecture

The architecture for BLE is split into three different parts: *controller*, *host*, and *applications*. The controller is usually a device implemented in hardware, that can transmit and receive radio signals. The host is usually a software stack that administers how devices communicate with one another and how services are provided. The application layer uses the software stack API to enable application development with the lower parts. The architecture is illustrated in Figure 2.5.

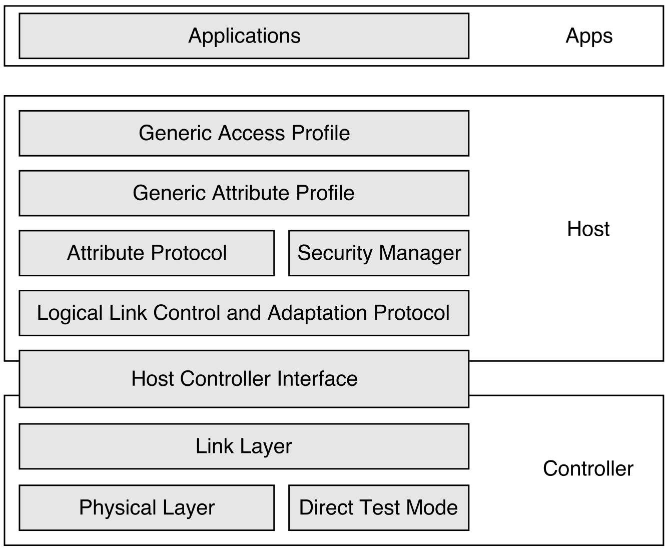

Figure 2.5: The Bluetooth Architecture.

2.6.1.1 Controller

The controller contains both analog and digital parts of the radio frequency components and hardware to support the reception and transmission of packets. The controller controls everything in between the antenna and the Host Controller Interface (HCI). The controller includes the Physical layer, the Link layer and communication with the HCI.

Physical layer

The Physical layer is responsible for receiving and transmitting bits using the 2.4GHz radio.

Link layer

The Link layer is responsible for advertising, scanning, and creating and maintaining connections. The Link layer provides two types of channels: *advertising channels* and *data channels*. The advertising channels are used by devices that are not in a connection. The data channels are used by devices once they are in a connection with another device. BLE use 40 channels, where channels 0 - 36 are data channels and 37 - 39 are used for advertising. Every advertisement packet is transmitted on each of the three channels. This minimizes the chance of collision with other advertisers.

The data channels are used through an adaptive frequency-hopping engine to ensure robustness, i.e., counteract packet loss.

Host Controller Interface

The Host Controller Interface (HCI) provides a standardized interface that allows a host to communicate with the controller. It allows a host to send commands and data to the controller and the controller to send events and data to the host. The HCI is composed of two separate parts: the logical interface and the physical interface.

Commands and events together with their associated behavior are defined in the logical interface. The logical interface can be delivered over a local API on the controller, where the API is defined by an embedded host stack included within the controller. It can also be delivered over any of the physical transports.

The physical interface defines how the commands, events, and data are transported over different connection technologies. USB, SDIO, and two variants of the UART are examples of defined physical interfaces.

2.6.1.2 Host

The Host is responsible for sending commands to the controller and receiving events back, and sending and receiving data from a peer device.

Logical Link Control and Adaptation Protocol

The Logical Link Control and Adaptation Protocol (L2CAP) component provides data services to upper layer protocols like the security manager protocol and the attribute protocol. The L2CAP is the multiplexing, i.e., combining multiple analog and digital signals into one signal over a shared medium, layer for BLE.

Security Manager Protocol

The Security Manager is responsible for pairing and key distribution. The Security Manager protocol provides cryptographic and data authentication functions.

Attribute Protocol

The attribute protocol defines a set of rules for accessing data on a peer device which is optimized for the small packet sizes used in BLE. The attribute protocol allows an attribute server to expose a set of attributes and their associated values to an attributed client. Attributes are addressed, labeled bits of data. An example of an attribute could be Temperature which data is written to by an application.

Generic Attribute Profile

The Generic Attribute Profile (GATT) is located above the Attribute Protocol. It introduces a number of concepts, e.g., *services* and *characteristics*.

The GATT describes a service framework using the Attribute Protocol for discovering devices and for reading and writing characteristic values on a peer device.

Generic Access Profile

The Generic Access Profile defines how devices discover and connect. It also defines bonding, which is how devices can create a permanent relationship.

2.6.1.3 Application Layer

At the top lies the Application layer. It defines three types of specifications: *characteristic*, *service*, and *profile*. These specifications is built on top of the GATT.

Profiles

Profiles are specifications that describe two or more devices, with one or more services on each device. A service can be used by many profiles to enable a given behavior on a device. Profiles can be seen as the embodiment of a use case or an application.

Services

Contrary to a characteristic, a service is a human-readable specification with a collection of related characteristics. For example, a heart rate service could encapsulate two characteristics: a mandatory heart rate measurement characteristic and an optional body sensor location characteristic.

Characteristics

A characteristic is a chunk of data that is labeled with a Universally Unique Identifier (UUID). Characteristics are defined in a computer-readable format instead of human-readable text.

An analogy might be a cabinet with drawers containing folders. The cabinet in this analogy is the GATT profile, the drawers are the services and the folders characteristics.

2.6.2 Advertising

Advertising is an important part of the BLE linked layer and essential for a device in order to be discovered, send data, and initiate connections.

For every advertisement, the device transmits the same packet on all of the advertisement channels (37, 38, 39) which is called an *advertising event*. The time spent on each advertisement channel depends on the data length. BLE's over the air data rate is as stated in Table 2.1: 1 Mbit/s, resulting in a transfer rate of 1µs per bit. A 100-bit long advertisement packet would thus result in spending 100 µs per advertisement channel.

A device will send an *advertising event* with a certain advertising interval. The advertising interval is thus the time between advertising events. The advertising interval can span from 20 ms to as infrequently as every 10.28 s [11].

There are five types of advertisements:

- **General Advertising**

  General advertising can be seen as the most general-purpose advertising type, it is basically a device saying "I want to be discovered". A device that performs general advertising can receive a connect request and go into a connection as a slave, it can also just be scanned by another device. A requirement for a device to perform general advertising is that it is not in a connected state.

- **Direct Advertising**

  If a device needs to connect to another device quickly, it can use direct advertising. To do this, direct advertising events are used. These packets contain two addresses: the advertiser's address and the initiator's address. A device cannot be actively scanned when using direct advertising and the direct advertising event cannot contain any additional payload.

- **Nonconnectable Advertising**

  Devices that does not want to be connectable use non-connectable advertising. A non-connectable advertising device will thus never enter the connection state. This could be used when a sensor that periodically broadcast its latest sensor data and has no other responsibilities.

- **Discoverable Advertising**

  Discoverable advertising is for devices that want to be discoverable, both for advertising data and scan response data, but do not want to be connectable. The advertising event contains an extended form of broadcast data, where the dynamic data can be included in the advertising packet and static data can be included in the scan response data.

  The broadcast data can be received by any active or passive scanning device nearby. The broadcasting device cannot know if any of the devices received its data since the broadcast data cannot be acknowledged. This makes discoverable advertising an unreliable option, which makes it unsuited for certain use cases.

- **Broadcasting**

  For a device to be considered a broadcasting device, the advertisement must contain some useful data. This excludes direct advertising devices from being broadcasting devices. Data is labeled within the advertising packets when broadcasting. An example of useful data can be the previously explained characteristics, i.e., broadcasting a service.

2.6.3 Scanning

Scanning is the method used if a device wants to receive advertising events. The device either scans for a connectable device or scans for data. Once a scanning device has received advertising data it can ask for more by sending a scan response request, resulting in that a device with support for scan responses sends additional data. The process is illustrated in Figure 2.6

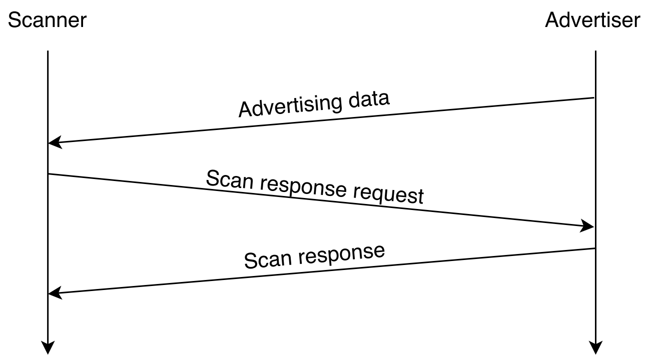

Figure 2.6: The communication between an advertiser and scanner asking for a scan response.

2.6.4 Universally Unique Identifier

A Universally Unique Identifier (UUID) is a unique number used to identify services, characteristics and descriptors also known as attributes. These UUIDs are broadcasted so that, e.g., a device can tell another device which services it provides. There are two types of UUIDs.
The first type is the 16-bit UUID, which is energy and memory efficient but due to the fact it only provides a limited amount of unique UUIDs there is a rule: you can only transmit the predefined Bluetooth SIG UUIDs [12] directly over the air. Because of this limitation, another type is needed.
The second type is the 128-bit UUID. This is the type you use when you want to create your own custom services and characteristics. You still define your custom UUID as 16-bits but it's incorporated in a *base UUID* which could look like this: *B131xxxx-7195-142B-E012-0808817F198D*, where *xxxx* is your own custom 16-bit UUID.

2.6.5 Whitelist

The effect of using whitelist is that packets from unknown sources would be filtered out at the link layer. Hence, minimizing the traffic in the host, allowing only packets from whitelisted devices reach the application layer. If used right, whitelist provides a security aspect and can also reduce the power consumption in the device.

2.7 Time Synchronization Standards

2.7.1 Network Time Protocol

The Network Time Protocol (NTP) [13] was first released in 1985 and is one of the oldest Internet protocols in current use. The Network Time Protocol is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks.

2.7.1.1 Architecture

The Protocol is used by all internet infrastructure and thus the most widespread synchronization technique. NTP uses a hierarchical, semi-layered system of time sources. The levels of this hierarchy is labeled *stratum* and is assigned a number where 0 is placed at the top (*stratum 0*), i.e., a server synchronized to a *stratum n* server will be running at *stratum n + 1*, see Figure 2.7 for illustration. NTP is based on UTC.

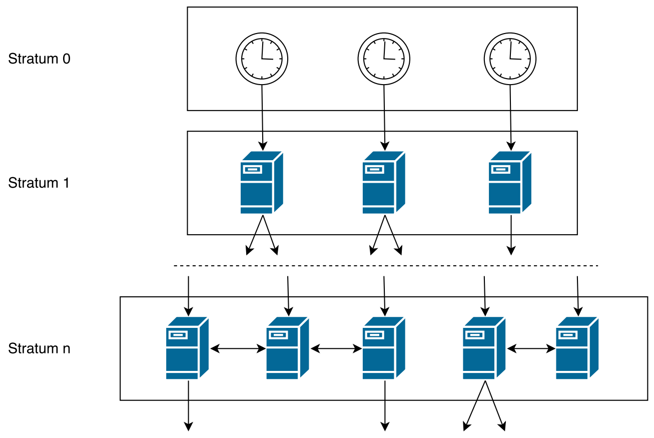

Figure 2.7: Illustration of stratum levels.

Stratum 0

Stratum 0 devices are also known as reference clocks. These are high-precision timekeeping devices such as atomic clocks or GPS clocks.

Stratum 1

Stratum 1 devices are also known as *primary time servers*. These devices are synchronized to within a few milliseconds of their attached stratum 0 device.

Stratum n

Stratum n devices are synchronized over a network to stratum n - 1 devices. A stratum n device may also peer with other stratum n devices to provide more stable and robust time for the peer group. The upper limit for stratum is 15, stratum 16 is used to indicate that a device is unsynchronized. So ideally you want your device to be as close to the reference clock as possible. The NTP algorithms on each device interact to construct a Bellman-Ford shortest-path spanning tree [14], to minimize the accumulated round-trip delay to the stratum 1 servers for all the clients.

2.7.1.2 Synchronization

A client synchronizes its clock to a server, i.e., a stratum at a higher level, by computing its time offset and round-trip delay. It is done by timestamping and sending packets, in cooperation with the server. The client starts by sending a request packet to the server and timestamps the leaving packet, t_0. The server timestamps the received request packet, t_1, and sends a response packet which also is timestamped on transmission, t_2. t_1 and t_2 is incorporated in the response packet sent to the client. At the reception of the response packet the client performs the last timestamp, t_3. The *Offset* is defined by

$$Offset = \frac{(t_1 - t_0) + (t_2 - t_3)}{2} \tag{2.2}$$

and the round-trip delay θ.

$$\theta = (t_3 - t_0) - (t_2 - t_1) \tag{2.3}$$

The values for *Offset* and θ are passed through filters and subjected to statistical analysis. The clock frequency is then adjusted to reduce the offset gradually and thus synchronizing with the server clock.

2.7.1.3 Availability

The NTP **reference implementation** is the original software implementation and has continuously been updated the last 20 years along with the protocol. It provides high accuracy timing, but may not be suitable for embedded devices with limited performance and memory space.

Simple Network Time Protocol (SNTP) is a minimalistic implementation of NTP. It uses the same protocol but without requiring the storage of state over long periods of time. It is preferably used in embedded devices. The downside is that the accuracy timing is worse.

2.7.2 Precision Time Protocol

PTP is a master-slave synchronization protocol and was originally defined in the IEEE 1588-2002 standard. The latest version, PTP version 2, was released in the updated standard known as IEEE 1588-2008 [4]. PTP is the go-to protocol if you need effective high precision time synchronization in a Local Area Network. The protocol was developed with ethernet cables in mind, but it has also been proven effective with 802.11 (WLAN) [15].

By its definition PTP is not an entirely software-based protocol, but rather relies on hardware based timestamping as close to the physical layer as possible.

Thus making the protocol sub-optimal for an arbitrary, Commercial Off The Shelf (COTS), embedded device without the hardware support needed.

Under the right conditions, PTP provides a standard method to synchronize devices on a network with sub-microsecond precision.

2.7.2.1 Architecture

The IEEE 1588 standards describe a hierarchical master-slave architecture for clock distribution, illustrated in Figure 2.8. PTP is based on International Atomic Time (TAI) [16].

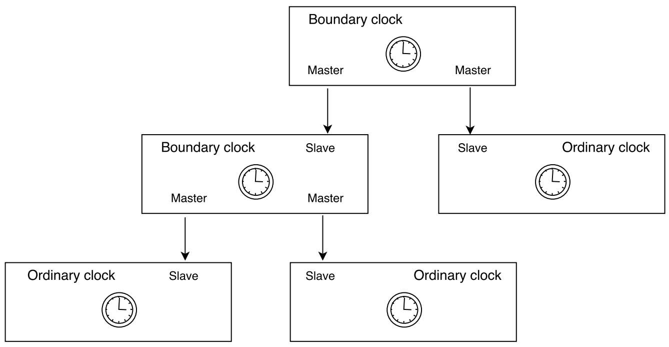

Figure 2.8: An example of Precision Time Protocol architecture.

2.7.2.2 Synchronization

There are three major steps for synchronizing devices using PTP.

1. Determine which device serves as the master clock.
2. Distribute the master clock time to all of its slave devices.
3. Measure and correct time skew between master and slaves caused by network delays and clock offsets.

The PTP uses the Best Master Clock algorithm [17] for deciding which clocks in the network that is the most precise and reliable (step 1). PTP timestamps messages to synchronize the clocks. These messages are called Sync and Delay_Req.

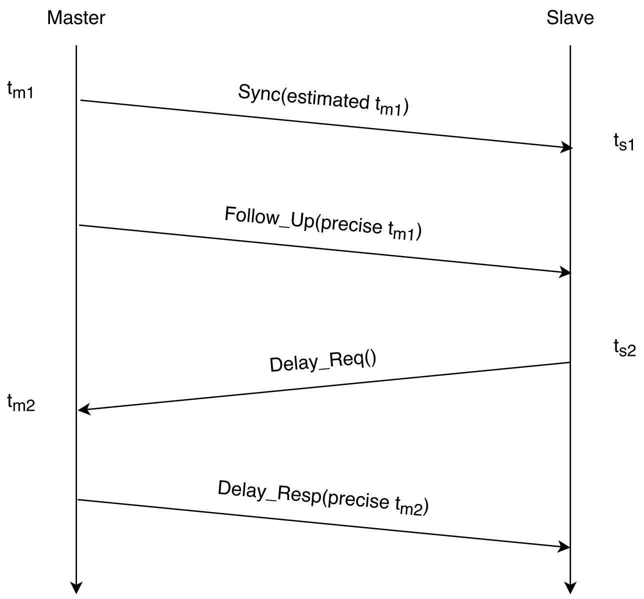

Figure 2.9: The Precision Time Protocol Synchronization Process.

The master sends the Sync message with a defined synchronization rate. The master clock timestamps the time, t_{m1}, when the Sync message is sent. The Sync message contains an estimate of t_{m1} and additional information about the clock used by the Best Master Clock algorithm in establishing master-slave hierarchy. When the Sync message arrives at a slave, it timestamps and stores the time of receipt t_{s1}.

The master then sends a Follow_Up message containing the actual value of t_{m1}, i.e., the exact time when the Sync message was sent.

Slaves periodically send Delay_Req and timestamps the transmission time, t_{s2}. The master precisely timestamps the incoming Delay_Req and stores the value in t_{m2}. The master then returns its observation in a Delay_Resp message. The synchronization process (step 2) is illustrated in Figure 2.9.

The slave now have enough data to calculate the master to slave delay d_{ms} and the slave to master delay d_{sm}

$$d_{ms} = t_{s1} - t_{m1} \tag{2.4}$$

$$d_{sm} = t_{m2} - t_{s2} \tag{2.5}$$

Using equations (2.4) and (2.5) it is possible to calculate an estimation of the one-way delay d_w and the offset of the slave clock with respect to the masters clock (step 3).

$$d_w = \frac{t_{ms} + t_{sm}}{2} \tag{2.6}$$

$$\mathit{Offset} = d_{ms} - d_w \tag{2.7}$$

The slave then adjusts its clock continuously to minimize the *Offset* value, thereby synchronizing with the master clock.

2.7.2.3 Availability

Hardware

PTP relies on hardware support to achieve high precision, and there is a big market with various product options that has embraced the technique. Everything from small embedded devices to server computers and network switches now has support for hardware based timestamping.

Software

When it comes to software implementation there are several open source alternatives. PTPd [18] is the most recognized project which is based on the IEEE 1588-2008 standard and to this date, continuously updated. PTPd is licensed under BSD Open Source License [19] and the authors of PTPd disclaims all liability since the standard may contain patented technology.

2.8 de facto Standards

2.8.1 Reference Broadcast Synchronization

Reference Broadcast Synchronization (RBS) [20] is a method in which receivers, i.e., sensor nodes uses broadcasts to compare clocks. This method differs from traditional methods, for example, NTP, which synchronizes the sender and the receiver clock.

The reference broadcasts do not contain an explicit timestamp, instead, receivers timestamp the arrival time as a point of reference for comparing their clocks, thus forming a relative timescale in the network.

The scheme was constructed to solve the issue of combining high precision time synchronizing with strict energy constrained devices.

The main strength of RBS is its broad applicability to commodity hardware and existing software in wireless networks. The main source of error is jitter in interrupt handling and decoding in the protocol stack.

2.8.1.1 Architecture

The motivation for changing the traditional synchronization method is the reduction of the critical path, i.e., the time it takes for a package to propagate through the sender's and receiver's network stack, Figure 2.10 illustrates the critical path of traditional time synchronization protocols.

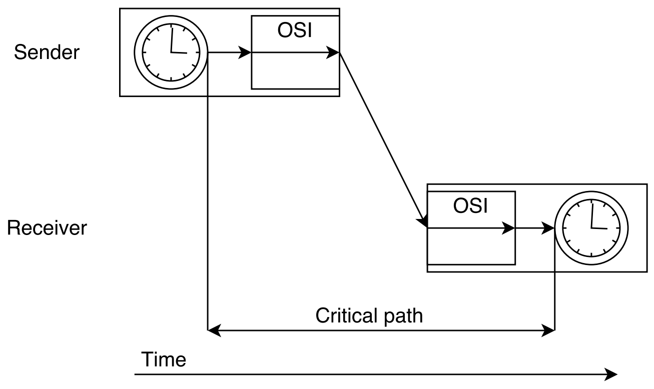

Figure 2.10: The critical path of traditional time synchronization protocols.

With RBS you remove the effect of Send and Access Time, just leaving the effect of the two error sources Propagation Time and Receive Time as illustrated in Figure 2.11.

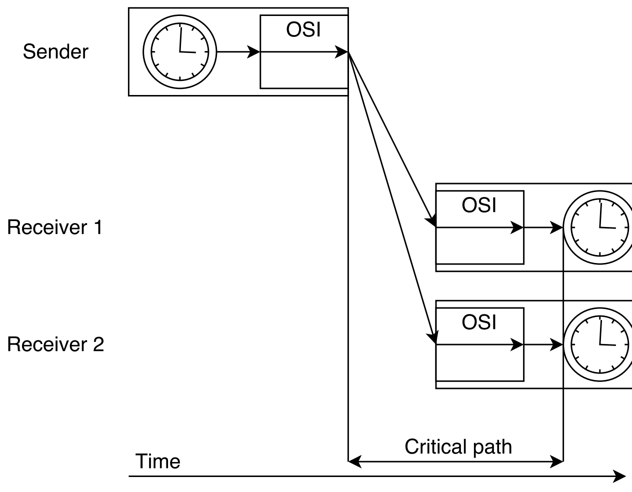

Figure 2.11: The critical path of RBS.

Propagation time, which can be seen as packet air time, will be considered 0 due to the fact that the propagation speed of electromagnetic signals through air is close to c^1 and short range networks rarely spans more than 20 meters, which would result in a maximum propagation time of 67 ns. This assumption leaves only the receive time as error source.

Studies also show that RBS has high energy efficiency and good scalability [21].

2.8.1.2 Synchronization

The synchronization process builds on sending broadcasts with a certain interval to the devices in a network.

Based on broadcasts, the receivers have enough information to form a local timescale in the network, which often is sufficient for sensor network applications.

By sending reference packets at a certain interval the precision of the synchronization can be statistically increased:

1. A transmitter broadcasts m reference packets
2. Each of the n receivers records the time that the reference was observed, according to its local clock.
3. The receivers exchange their observations.
4. Each receiver i can compute its phase offset to any other receiver j as the average of the phase offsets implied by each pulse received by both nodes i and j. That is, given
   n: the number of receivers
   m: the number of reference broadcasts, and
   $T_{r,b}$: r's clock when it received broadcast b,

[1]the speed of light in vacuum (299 792 458 m/s)

$$\forall i \in n, j \in n : Offset[i,j] = \frac{1}{m}\sum_{k=1}^{m}(T_{j,k} - T_{i,k}).$$

This basic scheme does not yet account for clock skew, i.e., the phase difference between two nodes' clocks will change over time due to frequency differences in the oscillators.

The RBS scheme use *least-squares linear regression* for finding the best-fit line through the phase error observations over time. Fitting a *line* to the data assumes that the phase error is changing at a constant rate, which is not true. This issue is solved by using a forgetting factor, so the best-fit line is based on a recent window of observation, i.e., forgetting data that is more than a couple of minutes old.

This scheme offers the ability to create a local timescale in a network, where each node knows its offset to every other node.

2.8.1.3 Availability

RBS has not had a major commercial impact on the synchronization scene and there are to our knowledge no open source implementations available. It is however widely cited in a lot of scientific papers and both TPSN and FTSP use ideas from RBS.

2.8.2 Time-sync Protocol for Sensor Networks

Time-sync Protocol for Sensor Networks (TPSN) [8] aims at providing network-wide time synchronization in a sensor network.

The algorithm works in two steps:

1. Establish a hierarchical structure in the network, this is the **Level Discovery Phase**.

2. Perform pair wise synchronization along the edges of this structure to establish a global timescale throughout the network, which is called **Synchronization Phase**.

Eventually, all nodes in the network synchronize their clocks to a reference node. TPSN relies on MAC-layer timestamping to minimize the source of error.

2.8.2.1 Synchronization

Level Discovery Phase

This phase runs once at the network deployment. The root node is assigned level 0 and initiates the level discovery phase by broadcasting a level discovery packet. The neighbors of the root assign themselves level 1. Then the level 1 nodes perform the same procedure to its neighbors. The phase is completed when all nodes are assigned a level. The network structure is tree type topology.

Synchronization Phase

The basic building block of the synchronization phase is the two-way message exchange between a pair of nodes. Propagation delay is assumed constant in both directions. Node A initiates the synchronization by sending a synchronization pulse packet at t_1. The packet contains A's level number, and the value t_1. Node B receives this package at $t_2 = t_1 + \delta + d$ where δ is the relative clock offset and d is the propagation delay of the pulse. At time t_3, B sends back an acknowledgement packet to A The acknowledgement packet contains the level number of B and the values t_1, t_2 and t_3. Node A receives the packet at t_4. The process is illustrated in Figure 2.12

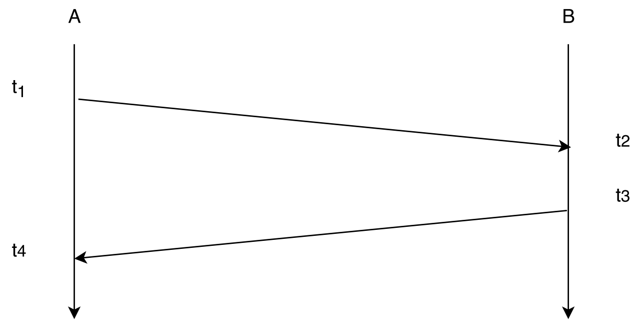

Figure 2.12: The TPSN synchronization process.

Assuming that the clock drift and the propagation delay does not change in this small span of time, A can calculate the clock drift and propagation delay as:

$$\delta = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{2.8}$$

$$d = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \tag{2.9}$$

Knowing the drift, node A can correct its clock accordingly, so that it synchronizes to node B.

2.8.2.2 Availability

There are no open source projects out there working on TPSN, but simple implementations can be found on, for example, `github.com` developed to execute on TinyOS [22]. It is not an available feature in the OS as FTSP is.

2.8.3 Flooding Time Synchronization Protocol

Flooding Time Synchronization protocol [23] (FTSP) is an ad-hoc, multi-hop time synchronization protocol for WSNs. It is similar to TPSN but sets out to improve some of the disadvantages of TPSN. FTSP states to be "especially tailored for applications requiring stringent precision on resource-limited wireless platforms".

Through definition, it builds on MAC-layer timestamping and broadcast. The definition thus requires a custom made stack implementation to enable MAC-layer timestamping.

For error compensation, FTSP uses linear regression to reduce clock skew and keeping network traffic overhead low. These definitions give a hint of RBS influence.

FTSP supports multi-hop synchronization.

2.8.3.1 Synchronization

Radio broadcasting is used to allow synchronization of multiple receivers using just one radio message. The message contains the sender's timestamp which distinguishes it from RBS.

Using this technique, one broadcast provides a synchronization point for all its receivers.

To solve the problem of clock drift linear regression is used to identify trends of the global time compared to the local time in the node. The linear regression is based on data points received in the past, much like the method used in RBS.

2.8.3.2 Availability

FTSP is an available feature in TinyOS [22], which is an open source, event-driven, and modular OS designed to be used with networked sensors.

2.8.4 Broadcast Synchronization over Bluetooth

Broadcast Synchronization over Bluetooth (BSB) [24] is a method for synchronizing sensor nodes over Bluetooth using broadcast messages. As in RBS, the broadcast message is used as a synchronization point, making it possible for the listening devices to create a local time with the aid of reference broadcasts. The paper bases its development on timestamping in the HCI layer. It does not go through how they create the local time thoroughly, other than naming statistical analysis.

2.9 Berkeley Motes

MICA motes [25] are wireless modules used for research of low power wireless sensor networks. The MICA motes are developed to use with UC Berkeley's TinyOS. The reason for mentioning the modules is because TPSN, FTSP, and RBS were developed for it, which will be of interest in the comparison of performance later on.

The de facto standards mentioned are almost ten years old, resulting in that they run the second generation MICA motes with the specs:

- 7.37 MHz processor
- 4kB of RAM

- 128 kB of flash memory
- 250ns clock resolution.

2.10 Least-Squares Linear Regression

The relative phase offset is affected by the nodes' clock skew. To model their skew, RBS and FTSP use least-squares linear regression to fit a line with minimal error to the relative offset between two nodes over time.

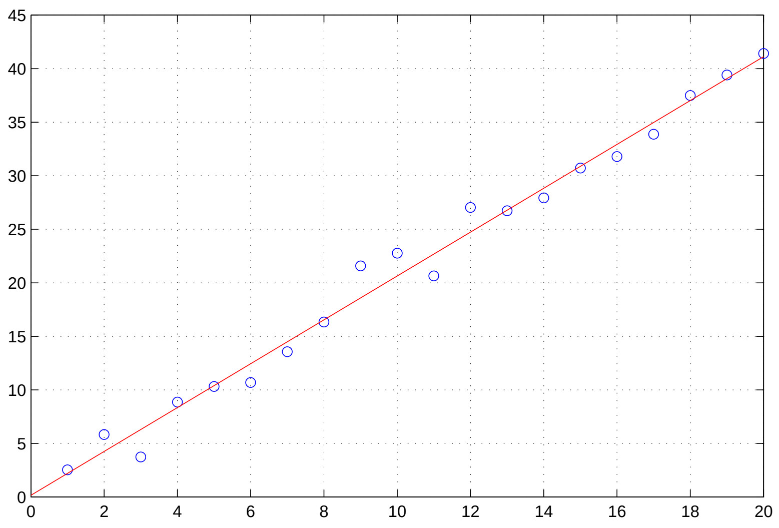

Figure 2.13: Linear regression: An example of best line fit used by FTSP and RBS

The fitting of a line assumes the clock skew to be constant. This is not the case, and the linear regression model has to forget data points older than a few minutes. Figure 2.13 illustrates how a line relates to a random number of data points using least-squares linear regression.

Chapter 3

Methodology and Tools

Before starting the case study of implementing a time synchronization algorithm for a wireless sensor network, some knowledge had to be gained. Articles and conference publications on the subject have been read and analyzed. The main target of this thesis has been to investigate what standards or de facto standards are available for getting a node, in a wireless network, to have the same perception of time as every other node in this network.

3.1 Case Study

The following sections will describe the implementation of a time synchronization method on u-blox Bluetooth low energy module NINA-B1, the tools used, and the decisions made during the case study. Since there have not been as many studies made on this topic of time synchronization using BLE, as there have been to other protocols. The case study was focused on using BLE as communication media between nodes within the wireless network.

3.2 Hardware

3.2.1 NINA-B1

u-blox latest short-range radio communication product is the NINA-B1 module. It is a small, stand-alone Bluetooth low energy module. The module uses the nRF52 chip from Nordic Semiconductors. The nRF52 is a low power system on chip (SoC). It uses an ARM Cortex-M4 processor and has a built-in transceiver that supports Bluetooth low energy. The nRF52 has a CPU clock speed of 64 MHz, 64 kB RAM, and 512 kB flash storage [26]. The NINA-B1 module can be powered by a 3.6V button cell battery and can last for years, depending on the application it runs [27]. The focus has been to implement RBS using several NINA-B1 modules.

3.2.1.1 Software Development kit

Applications running on the NINA-B1 module are written in C and interfaces the nRF52 chip with Nordic's SDK. This development kit is free to download from Nordic Semiconductor's website[1]. Included in the SDK are drivers, libraries, proprietary radio protocols, example code, and a complete Bluetooth stack called SoftDevice. Nordic describes their SoftDevice as [28]:

> "SoftDevices are precompiled into a binary image and functionally verified according to the wireless protocol specification so that all you have to think about is creating the application. The unique hardware and software framework provide run-time memory protection, thread safety, and deterministic real-time behavior. The Application Programming Interface (API) is declared in header files for the C programming language. These characteristics make the interface similar to a hardware driver abstraction where device services are provided to the application, in this case, a complete wireless protocol."

The Nordic's Softdevice does not expose any APIs to access the layers beneath the application layer. This made driver based timestamping impossible during the case study. Neither does the NINA-B1 module support hardware-based timestamps. The SoftDevice S132 version 2.0.0 was used during the case study.

3.3 Software Tools

3.3.1 Keil μVision IDE

The Keil μVision IDE is a software development solution from ARM that contains an Eclipse-based compiler, debugger, and project manager. During the case study, Keil μVision IDE was used to edit, compile, flash, and debug our implementation of a time synchronization algorithm.

3.3.2 nRF Master Control Panel

nRF Master Control Panel is an application for Android and IOS smartphones. This application lets the smartphone scan for, and connect to any Bluetooth device. During the case study, the application was used for debugging purposes [29].

3.4 Implementation Overview

Since timestamping in layers below the application is not possible in the NINA-B1 module, and the fact that a shorter critical path makes the time synchronization more precise, we choose RBS as the method to work with. RBS shortens the critical path by not transmitting any timestamps from the broadcaster. Thus removing the uncertainties from the sender.

A WSN that uses RBS as time synchronization method has the possibility to be extended without almost no effort. Its energy efficiency, and no need for network infrastructure (e.g. routers). The algorithm also eliminates a lot of uncertainties when it comes to broadcaster-receiver propagation time.

[1]www.nordicsemi.com

The WSN only uses Bluetooth discoverable advertisement to send synchronization information between nodes. In the case study, a minor modification of RBS was done to further decrease the power consumption of each sensor.

The system built during the case study consisted of three applications: *Beacon*, *Sensor*, and *Server*, and all run on separate NINA-B1's. In this section, not only the hardware setup and how the applications communicate is shown, but also how the algorithm is verified.

3.4.1 Hardware Setup and Communication

In Figure 3.1, the setup of nodes are shown. The beacon node is responsible for broadcasting the synchronization packet, which is done at a predetermined interval. The main task of the sensor nodes is to collect data from its surrounding environment. The server node is used to determine the system-global time of data collected by the sensor nodes.

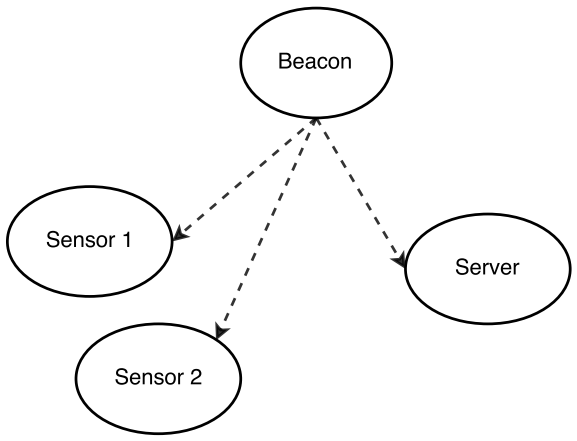

Figure 3.1: The hardware setup. The dashed arrows indicates wireless transmission of a synchronization message.

3.4.2 Beacon

The beacon has a service registered with a 128 bit UUID. This is the *synchronization service*. The beacon advertises its service at a given interval with a non-connectable advertisement. The advertisement payload is an ID telling which synchronization message it is. This device's only responsibility is to provide the other nodes in the network with reference packets.

3.4.3 Sensor

The sensor nodes are the ones collecting data from their surrounding environment. They passively scan for advertisements with a synchronization service. Once they receive a synchronization message, they take a timestamp from their RTC.

Each sensor also has a custom 128-bit service called *reference time service*. This service is advertised using discoverable advertisement. The advertisement data contains the sensor's timestamp of when the latest synchronization packet was received and which synchronization message it corresponds to.

3.4.4 Server

The server performs active scanning for synchronization and reference time services. If an advertisement containing a synchronization service is received, the server takes a timestamp from its RTC and notes the ID, exactly as a sensor does. If on the other hand, an advertisement containing a reference time service is received, the server reads the ID and the sensor's timestamp and calculates the offset to the value of its timestamp with the same ID.

3.5 Verification

To verify that the time synchronization algorithm works. A simple electrical circuit with a button was built to generate interrupts at a sensor. This device was connected to all sensors (Figure 3.2). The circuit generates interrupts simultaneously on all sensors. The sequence presented in Figure 3.3 happens for every sensor connected to the circuit. The same sequence is used to generate our results presented in Chapter 4, with the modification of a timer that generates a button press every 22 seconds.

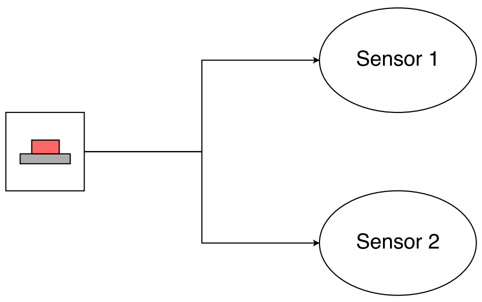

Figure 3.2: Illustration of the experimental setup: Pressing a button generates interrupt on all sensors concurrently.

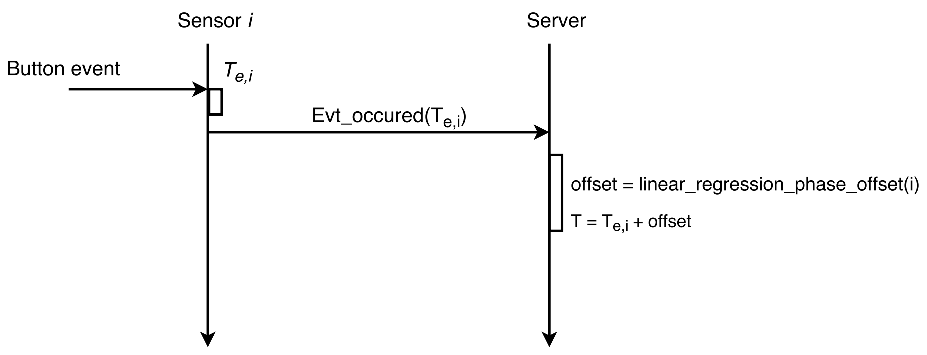

Figure 3.3: Experimental setup verification: At the press of a button, an interrupt routine is run. In this routine, a timestamp $T_{e,i}$ is taken and returns. Later, the timestamp is sent to the server where it is processed.

Chapter 4

Results

4.1 Literature study

The work performed included a literature study where different time synchronization standards and de facto standards were studied and compared. This section contains the result that was obtained from research of articles covering time synchronization over wireless short range networks. The results are highly dependent on the transfer medium, CPU, clock resolution, etc., making the results hard to compare. Papers concerning time synchronization scheme comparison has been taken into consideration.

4.1.1 Network Time Protocol

The NTP clients are said to synchronize their clocks to the NTP time servers with accuracy in the order of milliseconds using Ethernet (IEEE 802.3). In WSNs, non-determinism in transmission time can introduce several hundreds of milliseconds delay at each hop [21]. NTP also show limitations for WSNs in terms of energy and computation resources needed. Simple Network Time Protocol (SNTP) which is a minimalistic version of NTP solves these issues somewhat, but at the expense of accuracy. Experiments have shown that SNTP can achieve a time synchronization accuracy of 1 ms, under certain limited conditions, e.g., using a wired sensor network [30].

4.1.2 Precision Time Protocol

The IEEE 1588 Precision Time Protocol is the go to standard if high accuracy is the main goal. Though developed with Ethernet in mind it has also been proven efficient over 802.11 delivering a mean error of 7.35 µs [15]. The high accuracy is dependent on *hardware based timestamping* which puts some requirements on the devices used in, e.g., a sensor network. This requirement excludes the vast majority of commercial off the shelf (COTS) devices since they only support *application based timestamping*. A workaround for this issue can be to enhance the devices with an external timestamper which accurately timestamp interrupt signals marking the arrival or departure of a packet. This solution has proven quite efficient reaching a sub-µs accuracy [31] over Ethernet, using standard COTS devices. The accuracy is highly dependent on hardware, making PTP accuracy difficult to compile into a number.

4.1.3 Reference Broadcast Synchronization

The Reference Broadcasting Synchronization [20] scheme did as far back as 2002 deliver a mean error of 6.29 µs over 802.11. The scheme's main advantage is its scalability and energy efficiency.

4.1.4 Timing-Sync Protocol for Sensor Networks

The scientific paper [8] where TPSN originated claims an average error of 16.9 µs and a worst case error of 44 µs. In [8], they claim to have implemented RBS using the same equipment, for best comparison, resulting in an average error of 29.13 µs and worst case error of 93 µs. So according to their experiments, TPSN achieves almost double the precision compared to RBS.

4.1.5 Flooding Time Synchronization Protocol

FTSP [23] presents an average error of 1.5 µs on devices running TinyOS. The performance is significantly better than TPSN and it is claimed that the de facto standards have been running under the same conditions as in [8], making the results comparable.

4.1.6 Broadcast Synchronization over Bluetooth

The BSB [24] results in an average error of 4.56 µs with a worst case error of 17.4µs. The experimental setup consisted of a Bluetooth module controlled by a microcontroller with a 200 ns clock resolution.

4.1.7 Comparison

When comparing the NTP and PTP standards, it is clear that PTP achieves higher precision, just how precise is hardware and transmission dependent.

A survey conducted by North Dakota State University [21] has compiled the results from different research articles concerning WSN time synchronization schemes. The results are shown in Table 4.1, where we picked out the de facto standards interesting for this report. It is worth mentioning that all results derived from using 802.11 and Berkeley Motes.

Table 4.1: de facto Standards and how they perform

de facto Standard	Accuracy	Scalability	Energy efficiency
RBS	29.1 µs	Good	High
TPSN	16.9 µs	Poor	High
FTSP	1.48 µs	Average	High

The accuracy is in terms of *average error*.

4.2 Case Study

This section presents the results obtained from the case study. The effect of clock drift compensation, advertisement interval, whitelisting, and the number of nodes present in the WSN is illustrated with the aid of statistical analysis with the purpose of giving a good picture of the decisions made.

4.2.1 Experimental Setup

To generate data under long periods of time we had to automate the verification process. Instead of generating an interrupt on the sensors using a physical button, an interrupt was generated every 22 seconds. A constant in all experiments is the presence of one beacon and one server. The number of present sensors and the *advertisement interval* of the beacon is variable. These two factors will be clearly stated in each experiment.

4.2.2 Clock Drift

Before presenting the results from our implementation, it might be of interested to show how two device's clocks are affected by the clock drift (see Figure 4.1).

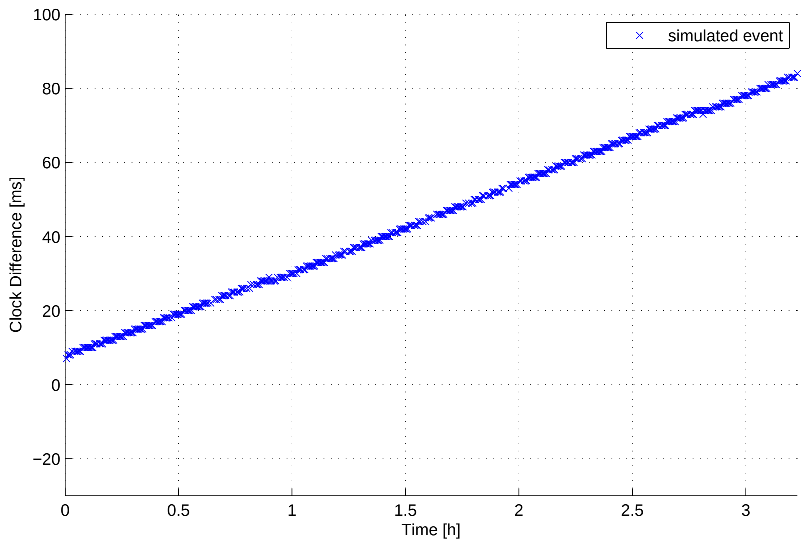

Figure 4.1: Clock drift: Illustrating how the error between two clocks grow without any time synchronization

By trying to start the two devices at the same time, we see an initial error of about 7 ms. During the three hours long experiment the error grew to about 80 ms.

4.2.3 Phase Offset Correction

The phase offset correction method proved efficient in the early stages of synchronization. As shown in Figure 4.2 the sensor's clock drift error had a great impact on the synchronization over time.

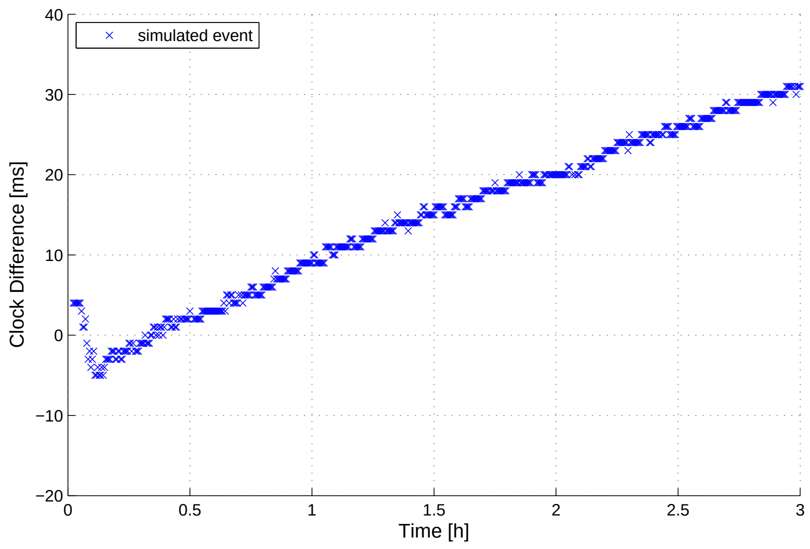

Figure 4.2: Phase offset correction: A collection of local time difference between two sensors.

The setup consisted of two sensors. The beacon broadcasted with an advertisement interval of 5 s. During the three hours long experiment the beacon broadcasted approximately 2160 advertisements.

4.2.3.1 Experimental results

Figure 4.2 clearly illustrate how the clock drift error effects the clock skew over time. Calculating mean error and the standard deviation was not done due to the obvious growing error.

4.2.4 Introducing Clock Skew Correction

To compensate for the clock skew a method involving least squares linear regression was introduced.

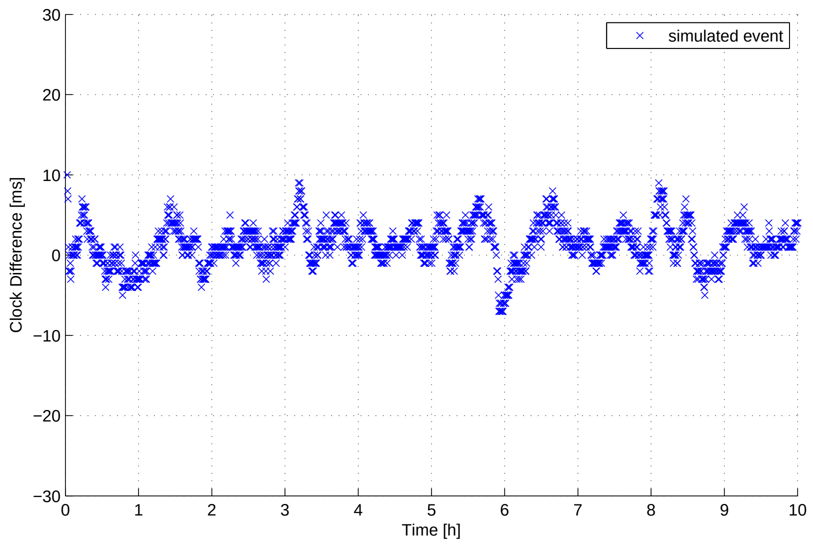

Figure 4.3: Compensating clock skew: A collection of local time difference between two sensors.

As can be seen in Figure 4.3 the method proved efficient, eliminating the effect of long term clock drift, clearly present in Figure 4.2.

The setup consisted of two sensors. The beacon had an advertisement interval of 5s. So during the ten-hour long experiment, the beacon broadcasted approximately 7200 advertisements.

4.2.4.1 Experimental Results

The experiment resulted in a mean error of *2.26* ms with a standard deviation of *1.76* ms.

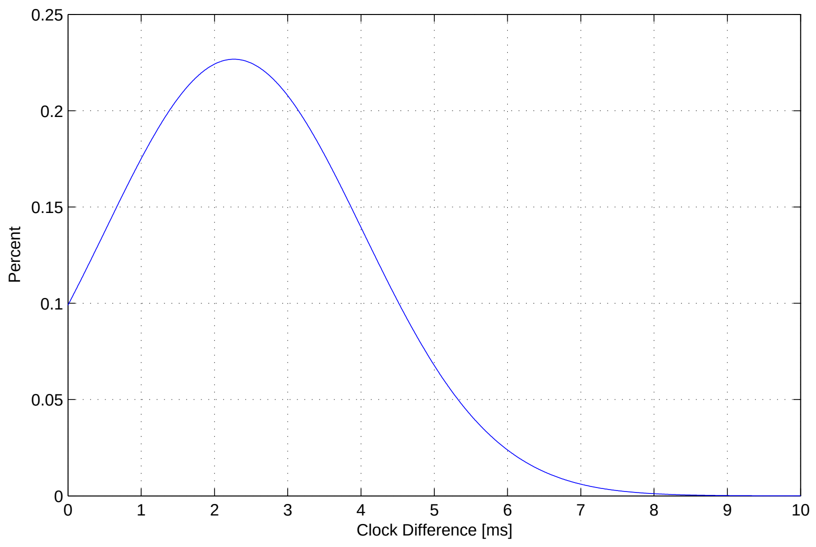

Figure 4.4: Distribution of the error providing linear regression.

The distribution of the error is illustrated in Figure 4.4.

4.2.5 Synchronization Intervals

In this section the effect of *advertisement interval* is presented.

Table 4.2: Synchronization Advertisement Interval

Interval	Accuracy	Standard deviation	Max error
2.5 s	2.31 ms	1.64 ms	8 ms
5 s	2.09 ms	1.74 ms	10 ms
10 s	4.37 ms	3.04 ms	13 ms

The accuracy is in terms of *average error*.

The setup consisted of two sensors. The beacon broadcasted with the advertisement intervals 2.5 s, 5 s, and 10 s. The effect of the different advertisement intervals is summarized in Table 4.2.

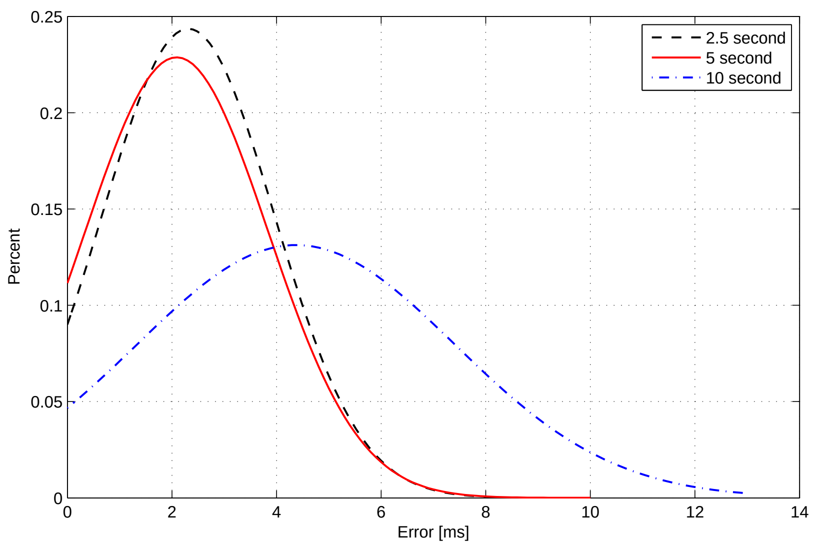

Figure 4.5: Effect of changing the advertisement interval. The error distribution for each of the three advertisement intervals.

The data in Table 4.2 and Figure 4.5 was derived from running the test case three hours each.

4.2.6 Network Load

The environment where the tests were performed experiences a heavy network load. To relieve the stack of all packets present in the network, whitelisting was tested. In the setup, two sensors were present. The beacon broadcasted with an advertisement interval of 5 s.

Table 4.3: Whitelisting beacon and sensors

Whitelist	Accuracy	Standard deviation	Max error
No	2.09 ms	1.74 ms	10 ms
Yes	2.14 ms	1.57 ms	11 ms

The result of introducing whitelist is presented in Table 4.3 and the effect on the normal distribution is presented in Figure 4.6, compared to a test run with the same setup without whitelist.

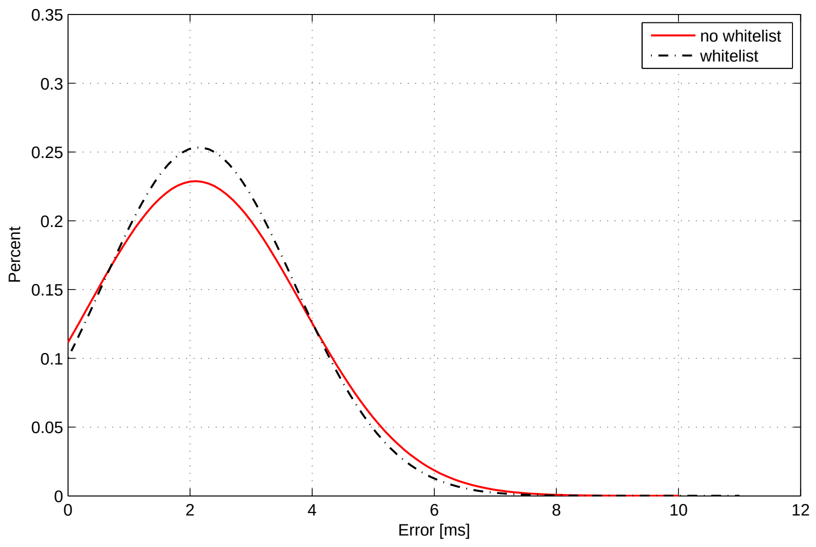

Figure 4.6: Normal distribution of the collected data using whitelist.

4.2.7 Introducing more Sensors

A wireless sensor network can contain everything from one to several hundred sensors. This section presents the result of introducing more than two sensors. The result is compiled in Table 4.4.

Table 4.4: Sensors

Sensors	Accuracy	Standard deviation	Max error
2	2.09 ms	1.74 ms	10 ms
3	3.99 ms	2.10 ms	16 ms
4	6.47 ms	2.45 ms	18 ms

This case used three different setups. The first WSN had two sensors present, the second WSN had three present, and the last WSN had four sensors present. The error on each event is calculated by subtracting the smallest timestamp from the biggest timestamp, thus providing the maximum error on each event. Figure 4.7 statistically illustrates the normal distribution of the different cases. The experiment was performed using an advertising interval of 5 s and each setup was running three hours.

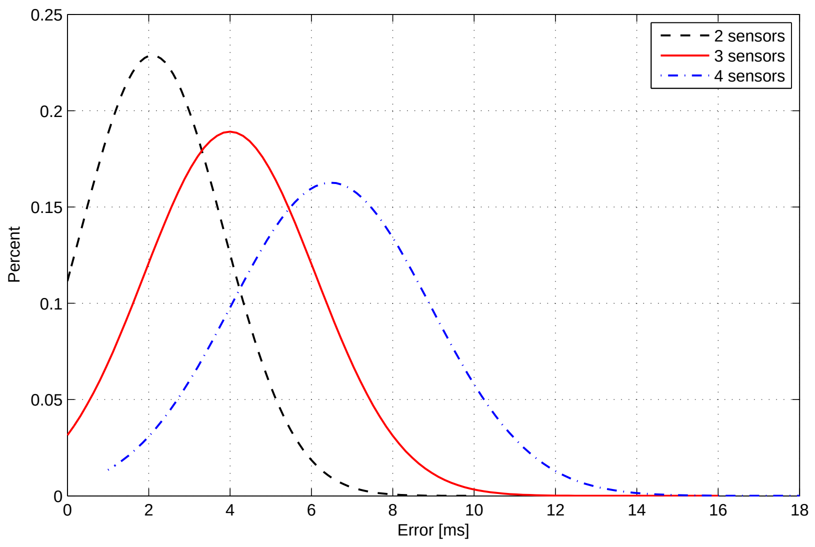

Figure 4.7: Normal distribution of the collected data using two, three, and four nodes present in the WSN.

4.2.8 Behavior over Time

The majority of experiments were performed during approximately three hours, resulting in unknown behavior if kept on for days. The results here were collected over a weekend, letting a WSN consisting of two nodes run almost 70 hours. The results derived from the data is presented in Table 4.5 and compared to experiments run under shorter time. The *advertisement interval* was 5 s.

Table 4.5: Comparison of behavior over time

Elapsed Time	Accuracy	Standard deviation	Max error
3 hours	2.09 ms	1.74 ms	10 ms
10 hours	2.26 ms	1.76 ms	10 ms
67 hours	1.83 ms	1.47 ms	10 ms

Chapter 5

Discussion and Conclusion

5.1 Discussion

For the case study, we implemented a time synchronization scheme influenced by RBS. The arguments for this decision was based on where we could perform the timestamping of arriving and departing packets. u-blox wanted us to perform the case study on a network consisting of their product NINA-B1 module, which only supports BLE and does not provide any hardware based timestamping or access to the host layer in the Softdevice. That left us with the option of application based timestamping. So the idea of removing the sender's non-determinism from the critical path seemed like the right approach, while we with this scheme could take advantage of BLE's energy efficient broadcasting technique.

5.1.1 Litterature Study

If the devices used has support for 802.11 and can perform hardware based timestamping, the PTP is recommended due to its accurate results and the access of constantly updated open source alternatives. When it comes to WSNs the de facto standards may be more aware of the energy and computation constraints. FTSP achieves the best accuracy among them and is featured in TinyOS which makes it an interesting candidate if the devices have the memory and computational possibility to run it.

RBS, TPSN, and FTSP are fair to compare due to the fact they have been tested using identical hardware and software. BSB, on the other hand, is not run on Berkley Motes, making the comparison a bit harder. For example in [20] RBS performed a mean error of 6.29 µs, but when implemented on Berkley Motes in [8] it performed a mean error of 29.1 µs, meaning the hardware has great impact on the result.

5.1.2 Case Study

5.1.2.1 Synchronization Interval

The results presented in Table 4.2 show minor difference in terms of average error between the 2.5 s and 5 s synchronization intervals. The implementation performs linear regression based on synchronization data that is only a few minutes old, meaning we forget the "outdated" data. This means that with the 2.5 s interval approach, the double amount of synchronization data was used in the linear regression model compared to the 5 s interval. The results show that storing more data, in this case, does not improve the accuracy. When increasing the synchronization interval to 10 s you see a change in terms of performance. The 10 s interval may be too large to recognize the minor tendencies in the clock drift, resulting in a mean accuracy approximately twice as big compared to advertising at a 2.5 s or 5 s interval.

5.1.2.2 Network Load

All experiments performed in the case study were done at the u-blox office. The Malmö office focus is short range networks, so extensive testing is always performed in the area, resulting in a high network load. It can be seen in Table 4.3 and Figure 4.6 that the effect of using whitelist did not enhance performance. This result raises the suspicion that the propagation time in the Host is not the main error source in our setup, i.e., the jitter in propagation time is not big enough to show any effect in the millisecond range.

5.1.2.3 Number of Sensors

The error grows with an almost constant rate per introduced sensor. Unfortunately, a lack of sensor nodes set a limit to the experiments, but the change in average error going from 2 sensors to 4 sensors proves that our implementation does not scale well. Due to memory constraints, we had to lower the amount of synchronization timestamps saved in the server when increasing the number of sensor nodes to 4, resulting in a more statistically unreliable linear regression, which could explain some of the increased uncertainty.

5.1.2.4 Behaviour over Time

The results in Table 4.5 shows that our implementation is stable over time, even showing a small fraction of lower average error compared to the three and ten hour-long experiments. There is no reason for the algorithm to be more precise when running longer but the result shows that it stays stable over long periods of time.

5.1.3 Error Sources

The results achieved did not reach near the same precision as most of the experiments in the concerned articles. This part is dedicated to error sources, to explain why our implementation did not achieve as a good result.

5.1.3.1 Application timestamping

Our approach is based on that all sensors timestamp reference points when receiving a synchronization service broadcasted by the beacon, but since we do it at the application layer the data has to propagate through the controller and host layers, i.e., receive time, which introduces non-deterministic jitter that is hard to foresee. NINA-B1 does not provide hardware support for timestamping, but to reduce the receive time effect on the timestamp it would be preferred to timestamp packets as low as possible in the software stack. A solution to this would be to not use Softdevice but rather implement a custom host. For example in the RBS [20] article, the introduction of timestamping closer to the physical layer resulted in a mean error of 1.85 µs compared to 6.29 µs when timestamping in the application layer.

5.1.3.2 RTC

The Real Time Clock in NINA-B1 uses a 32.768 kHz crystal oscillator which results in a tick resolution of 30.5 µs. The resolution is completely fine when using milliseconds as we did in our implementation, but if the time synchronization instead would want to achieve a sub-ms accuracy the clock would be a constraint. For example, the clock used together with the Berkeley motes had a tick resolution of 250 ns. The RTC in NINA-B1 is also used by the Softdevice which has the highest priority when it comes to usage, this could lead to nondeterministic delays in the timestamping of the synchronization advertisement and the simulated event.

Because of the amount of different tasks the RTC has to divide its attention to, it would be interesting to use an external clock to handle timestamping, and see possible results.

5.1.3.3 Advertising and Scanning

Since three channels are used for advertising in BLE and our implementation is so highly dependent on each device receiving the advertisement at the exact same time, the frequency hopping between the three channels will introduce non-deterministic jitter in the radio reception. The effect of the hopping scheme introducing non-deterministic jitter in radio reception contradicts the idea of deterministic receive time that RBS builds upon, which could have a great effect on the results. An advertising packet can be up to 31 bytes of data, resulting in 248 µs of transmit time on each channel. In the case where one sensor scans on channel 37 and another sensor scans on channel 39, the difference in radio reception would be almost 0.5 ms. A solution to this issue would be to only advertise and scan on one specific channel. Unfortunately the Softdevice API currently only support advertisement on one channel, but not scanning on one specific channel, resulting in too high packet loss ratio. It has come to our knowledge that using a feature provided by Nordic Semiconductors called Timeslot API [32] could enable scan on one specific channel, but due to lack of time we did not alter our implementation to fit with the Timeslot technique.

5.2 Conclusion

When it comes to time synchronization in short-range wireless networks, there is a wide range of available approaches. Depending on hardware and use case, the perfect standard or de-facto standard to use varies.

In this thesis we have shown that deploying time synchronization in a WSN using BLE is possible, reaching low-ms accuracy. Using BLE in a WSN may not be optimal in terms of performance, but it has great benefits in terms of energy efficiency. There is certainly room for improvement and we are sure that with some alternations and continued work, higher accuracy can be achieved. Timestamping close to the radio and a high-frequency clock is a common thread for successful time synchronization. The choice of a wireless standard may be a new property to take into consideration when constructing a WSN, where BLE introduces beneficial properties.

References

[1] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communixations of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[2] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *Journal of the Association for Computing Machinery, Vol. 34, No. 3. July 1987, pp. 626-645.*, 1987.

[3] D. L. Mills, *Network Time Synchronization: the Network Time Protocol on Earth and in Space.* Taylor & Francis, 2nd ed. ed., 2011.

[4] "1588-2008 - ieee standard for a precision clock synchronization protocol for networked measurement and control systems," 2008.

[5] J. Elson and K. Römer, "Wireless sensor networks: A new regime for time synchronization," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 149–154, Jan. 2003.

[6] "Real men program in c, http://www.embedded.com/electronics-blogs/barr-code/4027479/real-men-program-in-cnrf52832_ps_v1.0.pdf," Aug. 2016.

[7] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: A survey," *IEEE Network*, vol. 18, pp. 45–50, July 2004.

[8] Networked and Embedded Systems Lab (NESL), University of California Los Angeles, *Timing-synch Protocol for Sensor Networks*, (56-125B Eng. IV, UCLA EE Dept., Los Angeles, CA 90095), 2003.

[9] M. S. Gast, *802.11 Wireless Networks: The Definitive Guide.* O'Reilly Media, 2 ed., 2013.

[10] "Bluetooth specification version 4.2," December 2014.

[11] R. Heydon, *Bluetooth Low Energy: The Developer's Handbook.* No. ISBN-13: 978-0-13-288836-3, Prentice Hall, 1st ed. ed., 2013.

[12] "https://developer.bluetooth.org/gatt/services/pages/serviceshome.aspx," May 2016.

[13] "Network time protocol version 4: Protocol and algorithms specification," 2010.

[14] R. E. Bellman, “On a routing problem,” *Quarterly of Applied Mathematics 16*, 1958.

[15] T. Cooklev, J. C. Eidson, and A. Pakdaman, “An implementation of ieee 1588 over ieee 802.11b for synchronization of wireless local area network nodes,” *IEEE Transactions on Instrumentation and Measurement*, vol. 56, pp. 1632–1639, Oct. 2007.

[16] “Tai, http://www.timeanddate.com/time/international-atomic-time.html,” June 2016.

[17] J. C. Eidson, *Measurement, Control, and Communication Using IEEE 1588.* Advances in Industrial Control, Springer-Verlag London, 1 ed., 2006. pp. 42 - 47.

[18] “Ptpd, https://github.com/ptpd/ptpd,” May 2016.

[19] “Bsd open source license, https://opensource.org/licenses/bsd-3-clause,” June 2016.

[20] J. Elson, L. Girod, and D. Estrin, “Fine-grained network time synchronization using reference broadcasts,” *Proceedings of the 5th symposium on Operating systems design and implementation*, vol. 36, pp. 147–163, December 2002.

[21] P. Ranganathan and K. Nygard, “Time synchronization in wireless sensor networks: A survey,” *International Journal Of UbiComp*, vol. 1, pp. 92–102, April 2010.

[22] “Tinyos documentation, http://tinyos.stanford.edu/tinyos-wiki,” May 2016.

[23] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, “The flooding time synchronization protocol,” *SenSys '04 Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 39–49, April 2004.

[24] R. Casas, H. J. Gracia, Á. Marco, and J. L. Falcó, “Synchronization in wireless sensor networks using bluetooth,” *Third International Workshop on Intelligent Solutions in Embedded Systems*, pp. 79–88, May 2005.

[25] J. Hill and D. Culler, “A wireless embedded sensor architecture for system-level optimization,” 2001.

[26] “nrf52832 product specification v1.0, http://bit.ly/2909zsm,” Feb. 2016.

[27] “Nina-b1 product summary, http://bit.ly/1sqvs8o,” June 2016.

[28] “Softdevices, http://bit.ly/1uxyvtn,” June 2016.

[29] “Master control panel from nordic semiconductors, https://www.nordicsemi.com/eng/products/nordic-mobile-apps/nrf-master-control-panel-application,” June 2016.

[30] M. Ussoli and G. Prytz, “Sntp time synchronization accuracy measurements,” *IEEE 18th Conference on Emerging Factory and Automation*, pp. 1 – 4, September 2013.

[31] A. Mahmood and T. Sauter, "Improving the accuracy of software-based clock synchronization and encountering interrupt coalescence," *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pp. 82 – 87, October 2015.

[32] "Softdevice specification: S132 softdevice v2.0," June 2016.