Investigating the Feasibility of Time Synchronization in Bluetooth Low Energy Networks: Protocol Design and Experimental Validation

Merlijn Siffels - University of Twente

July 2023

1 Abstract

Bluetooth Low Energy (BLE) has gained widespread adoption as the wireless protocol for Internet of Things (IoT) devices, encompassing various domains such as medical, personal, and technical applications. To facilitate specific applications that rely on a shared and accurate timebasis, the research and testing of a shared timebasis protocol for BLE has been undertaken. The primary objective of this thesis is to design and research the possibility of a Bluetooth Low Energy time synchronization protocol capable of being used at BSS in their pain research. Having a synchronized timebasis will allow for faster analyzing of evoked potentials in EEG measurements. To validate the effectiveness and possibilities of a protocol, experimental testing is conducted, evaluating performance of the Silabs Explorer kit in terms of synchronization accuracy, and reliability. The results obtained will provide valuable insights into the feasibility of time synchronization in Bluetooth Low Energy networks. The suggested protocol is divided in 3 stages. A connection phase, where the offset of the internal timers is corrected. A start-up phase, where the expected drift is measured and extrapolated to adjust the internal timers. And an active Resync phase, to keep the internal timers in sync indefinitely.

Contents

2 Introduction

This paper will be about the development and implementation of a time synchronization protocol in Bluetooth low energy(BLE). At the University of Twente the BSS department is engaged in pain research. In this research they provide an electrical stimulus through a device called an Ambustim. After providing this electrical stimulus they measure the brain response using an EEG. Currently the setup is wired, and they are moving to a wireless setup to remove potential points of failure. This setup can be seen in Fig: 2. A problem that arises by moving to wireless setup is the fact that it will become uncertain when certain actions have taken place relative to other devices in the wireless network. This is due to the fact these devices all have their own internal timer, with their own characteristics. The new setup will be controlled from a central server, a computer, and will have multiple peripheral devices, the Ambustims and the EEG. Through the measurement of multiple brain responses a signal can be derived. It is essential for the EEG signals to know when the stimulus was provided, as the signal should be overlaid. If there is an offset of when the response is expected, relative to the actual stimulus, will cause the brain response to be measured as noise, instead of as the signal. This can also be seen in Fig:1.

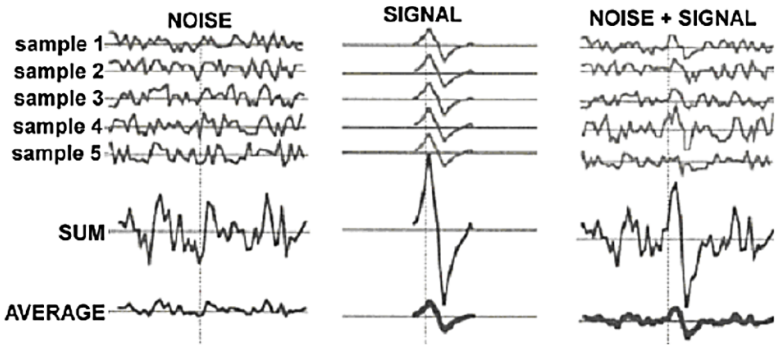

Figure 1: Example of how overlaying multiple measurements from an evoked potential can result in deriving a signal, while from one measurement deriving an accurate signal would be improbable. [1]

Thus to avoid this a time synchronization protocol is developed with the goal of a maximum time synchronization drift relative to the server timer of 500 μs. This is determined by the sample rate of the EEG, the EEG makes a sample every 500 μs.

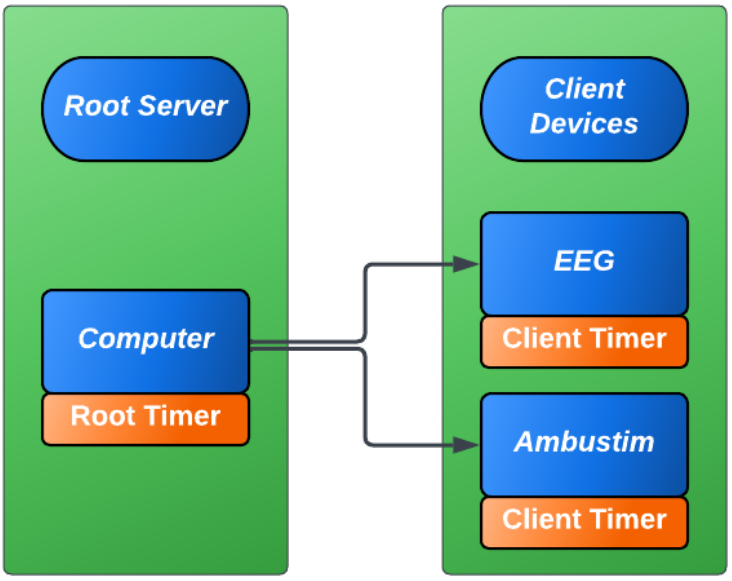

Figure 2: Overview of the Setup of BSS. A computer which acts as a central server, with client devices which connect with this computer. The timers of these client devices follow the timer of the root server.

2.1 What is Time Synchronization

In the context of this thesis, time synchronization refers to the coordination of actions performed on wireless devices, ensuring that these actions occur simultaneously when observed from an external perspective. It entails aligning the internal clocks of different devices to a shared time basis, enabling precise coordination and accurate timing of events. With time synchronization, if a specific action is intended to occur at a particular moment, when it happens synchronously across multiple devices, this means that the devices are synchronized in time. In the context of pain research it will allow for the accurate overlaying of evoked potentials to derive a brain response. This is because the moment that a stimulus was provided is known by the EEG and the Ambustim, which provides trigger points of evoked potentials.

2.2 Barriers to Time Synchronization

There are 2 elements that makes wireless devices not synchronized, this is the offset and crystal drift. Each wireless device has an internal timer. The first problem arrives from the offset between 2 different timers. When a device is turned on their internal timer will start counting, it is unknown to this device

when another device was turned on. While one device might have their internal timer at minute 5 of operation, a newly turned on device will start counting from 0. This is the offset between devices which has to be resolved. When this offset is resolved the second barrier to time synchronization becomes apparent, this is the timer drift over time. Internal timers, when compared to a reference timer, will drift away from this timer. To understand this what must first be discussed are the crystals that are used by devices. Crystals are used for their ability to vibrate at a certain frequency. This frequency is used as a clock for actions to be performed on these devices. These crystal are accurate, but not perfect. It is best explained with an example. A crystal might be assumed to have a frequency of 1 MHz. However, in production it is not possible to guarantee that the crystal will vibrate at this exact frequency. It might vibrate at a frequency of 0.999995 MHz. This is a deviation of 5 parts per million from the advertised value, and will result in timers that are based on this crystal to drift away from a 1 MHz crystal timer at a rate of 5 μs per second. Over an operation of one hour this results in a drift of 18 ms.

2.3 What is Bluetooth Low energy

Bluetooth low energy is a wireless protocol that is increasingly becoming more popular in the Internet of Things. It is more flexible than Bluetooth classic in regard to protocols. Where Bluetooth Classic has a call robin protocol where it communicates with up to 7 nodes. BLE organizes communication through connection intervals. In BLE the protocols can and often are designed by the developer for a specific application. Another difference between BLE and Bluetooth classic is the power consumption of the wireless communication. Bluetooth low energy consumes less power compared to Bluetooth Classic which makes BLE the preferred choice for most portable battery applications, especially medical devices, use BLE over Blueooth Classic. However, with the drop of power consumption BLE has a lower data transfer rate compared to Bluetooth classic. [2]

2.4 Terminology

In order to have firm grasp on BLE protocols it is important to define the terms used in the paper. In BLE there are 2 roles. These are the slave and master roles, which are now known as the central an peripheral role. A peripheral device, is a device that provides data or services to other devices in the network. It typically operates in the advertising mode, where it broadcasts its presence and available services. Peripherals can be sensors, actuators, or any device that generates or collects data to be used or consumed by other devices. A central device in BLE is a device that initiates and manages connections to peripheral devices. It scans for nearby peripherals, establishes connections with them, and can request or receive data or services from the connected peripherals. Central devices are usually smartphones, tablets, or other devices that interact with and control peripherals.

Besides Central devices and peripherals there are other terms of BLE with a different functionality. These are the Server and Client roles.

The server is a computing system or application that provides services, resources, or data to other devices or applications. It is responsible for processing requests from clients and responding with the requested information. The server hosts the services or data and waits for clients to connect and make requests.

The client is a device or application that requests services or data from a server. It initiates communication with the server, sends requests for specific information or services, and receives the responses from the server. Clients can be computers, smartphones, IoT devices, or any device or application that interacts with and consumes the services provided by the server.

In most applications the central device is the client, while the peripherals are the server.

Another commonly used term is a connection interval. A connection interval, within the context of BLE, refers to a predefined time duration during which two BLE devices exchange data packets within an established connection. It represents the interval at which the central device and the peripheral communicate with each other. The ranges of a connection interval are typically between 20 ms and 100 ms.

2.5 Relevance of Time Synchronization

In the next section 2.6 the specific use cases and importance of having synchronized wireless devices defined in the context what this report will focus on. Other uses for a synchronized timebasis is to reduce power consumption. Certain medical devices sleep for prolonged times to conserve power. With an uncertain drift of the internal timer crystals when these devices wish to re-establish a connection it will cause for trouble. It forces the devices to be awake for a longer period of time to give leeway in case the timers have drifted greatly. Or the devices have to re-establish their connection periodically to ensure that the drift never becomes unrecoverable. With a synchronized timer that has corrected for this drift the system will be able to stay awake for a shorter time, or the re-connection period can increase. In both cases the power consumption is decreased with no drop in capabilities.

2.6 Use Cases

2.6.1 BSS-group UTwente - Actuator based network

At BSS BLE is used for the control of stimulation's. Then EEG data is measured and saved. These stimulation's are provided by Ambustim devices. A measurement session can have hundreds of stimulation's and brain responses. The timing and intensity of electrical stimulation's of the Ambustim is controlled through BLE. These controls are short data-packets are send infrequently, every second. There is a low demand of bandwidth for the application, the commands to the Ambustims, to run.

As stated in , the sample rate of the EEG is one sample every 500 μs. To get all measured response signals overlayed, the requirement of a maximum synchronization drift of one sample is derived. This will allow for the EEG to set trigger points of when a stimulation was provided. Using these trigger points and the hundreds of measurements will allow for the brain response to the stimulation to be quickly and accurately be processed. Through overlaying the measurements.

The entire setup at BSS will be controlled from a central computer which servers as the server and root timer. The internal timers of the Ambustim and the EEG will follow a protocol that makes them follow the root timer of the computer. These timers of peripheral devices will be called peripheral, or client timers. While the root timer is also called the central timer.

2.6.2 TMSi - Sensor based network

TMSi also implements a stimulator and EEG setup. The largest difference between the system of BSS and of TMSi systems is that TMSi is transmitting the EEG data through BLE, which creates a continuous demand of bandwidth. This limit of available bandwidth provides for a different challenge and environment compared to the setup of BSS. The nigh-guarantee of having a packet at the start of a connection interval provides an opportunity for time synchronization. To reduce the bandwidth usage of the protocol TMSi has suggested using fluctuations in length of the connection interval to estimate the drift between the root timer, and the peripheral timers. Using the fluctuation in length of the connection interval requires no unique synchronization data packets to be send, and thus this method requires no bandwidth to work. This principle will be explored in more detail in 3.2.

3 Existing BLE Synchronization protocols

BLE is a relative new wireless communication protocol. The is no time synchronization protocol that is yet common place. There are 2 major barriers to time synchronization that a system should overcome. These are the offset and crystal drift of timers.

3.1 Bluesync

Bluesync is an accurate time synchronization protocol built around a short startup time to estimate the crystal drift of different timers in the network. It experiments with different drift correction algorithms that are based on receiving a notification from the central clock periodically. The peripheral timers then measure their error compared to this central time and use this to create a drift correction algorithm. From Bluesync it becomes evident that having more measurements for drift estimation will reduce the error of the timers. The protocols are used are based on the average error or linear regression. With the startup

phase lasting as short as 800 ms and as long as 16 seconds. And it compares the results of the different implementations. From these tests it became evident that using a linear regression algorithm will reduce the error the greatest. As well as having more and longer intervals between measurements improve the estimated drift.[3]. This protocol does not have an indefinite synchronization, meaning that the longer the devices are operational, the greater the synchronization error between the devices. A major difference between Bluesync and the application of BSS is that the devices in Bluesync do not engage in a connection with each other. There is no data transfer between devices. And the client timers follow the timer of a wireless device that operates as a BLE beacon.

3.2 Synchronization in wireless biomedical-sensor networks with Bluetooth Low Energy

In Synchronization in wireless biomedical-sensor networks with Bluetooth Low Energy, also referred to as Bideaux, they attempted different methods of reducing the offset of timers. From Bideaux it becomes evident that using the connected event that is provides by the Bluetooth low energy stack can reduce the offset to be on average 40 μs [4]. This is the main motivation to use the connected-event to remove the offset. Bideaux does not have drift correction/estimation, and is solely focused on the initial offset. The other methods they researched provided worse and more unstable offset reductions.

4 Developed BLE-Protocol

In this section an indefinite time synchronization protocol is described and explained for the usage at the BSS department of Utwente. It consists of 3 parts. The connection phase, the Startup phase, and the Resync phase. The connection phase is used to adjust the offset, the Startup phase is used to estimate the timer drift, and reduce future required bandwidth, and the Resync phase is used to keep the system synchronized indefinitely.

4.1 Limitations

This section will go into the most significant limitations when trying to achieve time synchronization. This protocol was developed and tested on the Silicon labs explorer kit. When working with the BLE stack there are only 2 ways to interact with it. These are events and API. The BLE stack returns events, for example if a packets has been received, or is able to perform an action through and API. For example a read request. A visual representation is show at Fig: 4 One of the issues that becomes evident from building on the application layer is the fact that it is unknown when a connection interval starts, and when it ends. By not knowing this it becomes impossible to synchronize the timers by simply sending a packet and reading it. There is an unknown error that is introduced by the uncertainty of the connection interval. If the server timer is measured

and send to the buffer 50 ms into a connection interval, will need a different correction compared to when it would be send 1 ms into a connection interval. Furthermore, there are a few delays that are caused by the link layer. The sending, and processing of data. However, this delay is constant and mainly determined by the length of a data packet. Moreover, the Bluetooth module runs on an independent radio clock. However this value cannot be read. It is not possible to know the value of where the Bluetooth radio clock is at. Therefore a second timer has to be used in parallel to still allow for actions to be performed based on time passing. Due to the fact that the BLE is protected to ensure proper operation, it is not possible to use this parallel timer to force an interrupt and use a BLE function, for example a write request. Attempting to send data with this parallel timer will result in failure of the send and return an error code. However, this parallel timer is the most accurate timer available and needed for timestamping and time enstive operations. This forces the use of a timer that is compatible with BLE.

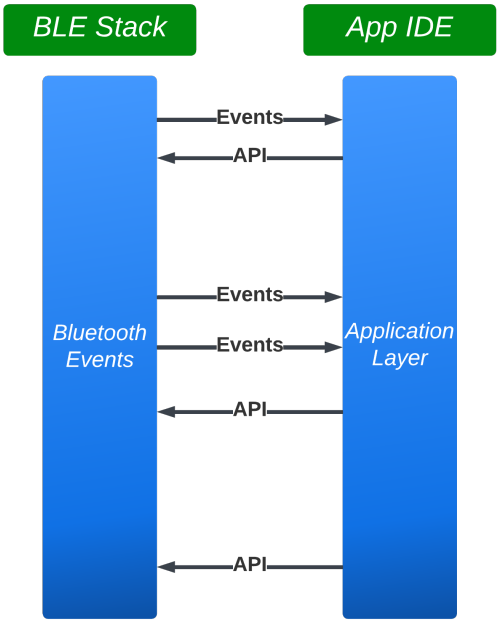

Figure 3: A representation of how to interact with the BLE stack. It is only possible to communicate with BLE through events and APIs.

4.2 Software and Hardware

Used Nordic NRF52 Development kit[5] and Silabs Explorer kit. [6] Nordic has a 32 MHz high frequency crystal. While Silabs has a 38.4 MHz high frequency crystal. The timers of the devices are controlled by this high-frequency crystal.

While these crystals are accurate, there is still a typical error. This is defined as ppm. An error of 100 ppm would result in a uncertain drift of the timer of 100 microseconds every 1 second. The high frequency system clock used in Silabs has a maximum drift of +- 10 ppm. And a typical drift of 5 ppm. On average it would take 100 seconds for the clock to drift 500 microseconds. This is too significant of a drift for prolonged operations. For all experiments only the Silabs explorer kit is used, except for the connected event experiment, for which both the Silabs Explorer kit, as well as the NRF52 Development kit are used.

4.3 Connection Phase

A method to remove the offset of the peripheral timers relative to the root timer has been developed. The focus of this method was for it to be scalable, and reliable. A quick summary of the problem is that internal timers will start counting at the moment they are turned on. However, to the wireless devices it is unknown when other devices in the network have been turned on. To remove this offset the connected event that is provided by the BLE-stack will be used. In Bideaux the initial offset is reduced to 32 μs or 64 μs. [4]. It is either of these 2 values due to the frequency of the clock that is used for timestamping. The clock has a frequency of 32 kHz. This means that for the 32 μs, one period has passed, and for the 64 μs 2 periods have passed. The devices used to explore this have timers with a frequency of 32 MHz, or 38.4 MHz. With a connection between one root node and one peripheral node it becomes possible to set both timers at 0 at the connected event. This would synchronize both timers to eachother. One issue that arises with this approach is that the system is no longer scalable, when a new peripheral node joins the network, the first peripheral node timer will be out of sync with root timer, as the root timer will reset to 0, while the first peripheral timer will keep counting. In the minimum setup for the BSS pain research there are 3 devices. The computer with the root timer. The EEG measurement device, and the Ambustim. To make the system scalable more logic is used. At a connected event the root timer and the peripheral timer will take a timestamp. The root timer will send the timestamp it took to the peripheral timer. This peripheral timer will use this timestamp, in combination with the elapsed time between when the connected event took place, until the packet was received, to correct the offset of its internal timer. This should set the peripheral timer to the root timer. This method is scalable, for every new connection will synchronize their peripheral timer with the root timer.

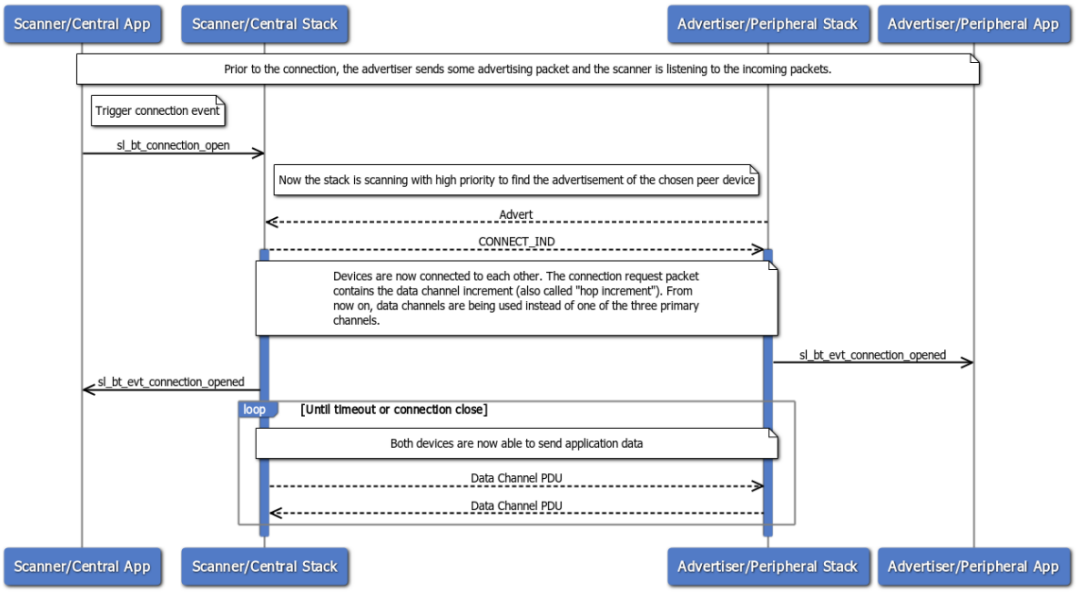

Figure 4: A diagram that displays the structure and behaviour that occurs to receive the connected event. [7]

4.4 Drift estimation phase

The second step to achieve time synchronization is accounting for timer drift, and creating a drift estimation protocol. Timer drift is caused by imperfections of the internal crystal. Furthermore, with changes in voltage/temperature the frequency of a crystal also changes. This means that manually calibrating for the imperfections of the crystals has to take these changes of environments into account. To create an automatic drift estimation 2 methods are researched and implemented. The first method is of interested to TMSi, it uses the principle of fluctuations in the connection interval to create a drift estimation. The length of a connection interval is known, for example 100 ms, if a peripheral node would measure that the start of a connection interval is every 101 ms, would mean that the timer of the peripheral node is running too fast. By observing these fluctuations over time a drift estimation is made. The second method is used in Bluesync [3], which is based on sending the root timer value to peripheral timers periodically. Through the expected period of time between sends, and the server timer value compared to the internal timer value, a drift estimation is made. What should be accounted for is the transmission time between sending this server timer value, and when it is received. Most significant of this transmission time is the time a data packet spends in the buffer of the BLE stack, before it is send. In BLE the data transfer method with the shortest transmission time is a notification. Which will be the packet type used in this implementation. Using the data from these notifications a drift estimation is made. Using multiple

datapoints improves the accuracy, as well as having a longer period between each datapoint improves the accuracy of the drift estimation. This is proven by Bluesync [3]. In this implementation an average error algorithm will be used to derive drift estimation.

4.5 Resync Packets

To keep the timers in sync indefinitely a resync phase is part of the protocol. The drift correction from the startup phase is based on a relatively small amount of data, a couple of seconds, which is then extrapolated. While research shows[3] that this can be accurate, over time the timer drift will keep growing. To achieve consistent time synchronization over long periods of time a re-synchronization protocol is implemented that runs parallel to the operation of the an application. This resynchronization will be based of notifications which send the root timer value. This will run throughout the entire operation of the application. The reason to have a separate start-up phase from the resynchronization phase is to reduce the bandwidth usage that is required during the application. The heavy lifting of the drift estimation is done during the start-up of devices, while corrections are performed during the resync phase. In the intended application of BSS and TMSi which consists of a computer which acts as a server and root timer, with multiple peripheral devices connected, the Ambustim and EEG, all devices will be turned on and during the operation no new devices will join. For applications where during the run of the application new devices join the network, are able to synchronize with the root timer and develop a drift estimation, but this will take longer, or will require for the resynchronization packets to be sent more frequently. The period of when the resync packets will be send change with different requirements. For the intended application of BSS, with crystals having a maximum ppm of 10, and a typical ppm of 5. A period 25 seconds for a resync packet is the maximum period recommended. This is to avoid the case where the timer drift will be greater than 500 μs during the operation. If the root timer has a crystal that runs too fast by 10 ppm, and the peripheral timer has a crystal that runs too slow by 10 ppm, this will cause a time drift error of 20 microseconds per second. In 25 seconds the timers will thus have drifted apart. This is an unlikely case, but to avoid a time drift of more than 500 μs the periodic correction of the timers should be at least every 25 seconds.

4.6 Adjusting Timers

The peripheral timers will follow the root timer, this is achieved by adjusting the peripheral timer values periodically. It is not possible to speed up or slow down the high-frequency crystal that these devices rely on. Another method has to be implemented. The method implemented in this protocol is adjusting the timer counter value periodically. To understand what this means it is important to know how the timers operate on the device. On the Silicon Explorer kit the timer is based on the 38.4 MHz high-frequency crystal. The code that runs on

this kit is able to access the timer through a counter block. This counter block value increments over time based on the ticks of the crystal and a prescaler value. This prescaler value determines how many ticks of the crystal is needed to increment the value in the counter block. For a prescaler of 32, this is 32 ticks. This effectively reduces the frequency of the crystal by 32. The value in the counter block can be read, or overwritten. By reading the value in the counter block it is possible to create time based applications, for example through using timer interrupts certain functions can be called, or values can be changed. For the drift correction application there will be a periodic interrupt which overwrites the value in the counter block. If the drift correction algorithm dictates that a timer runs too fast, then through overwriting the value in the counter block the timer can be made to run slower. For example, every second the value inside the counterblock can be reduced by 5 μs. The actual correction is based on the drift correction estimation.

Clock overflow Both the Silabs explorer kits as well as the Nordic development kit have 32 bits and 16 bits timers. When the value of the timer exceeds this bit value it will overflow. Depending on the frequency of the clock these timers will overflow at different rates. For example the Silabs Explorer kit has a timer based on a high-frequency crystal which has a frequency of 38.4 MHz. If it uses a 16 bit timer it will overflow every 1.7 ms. While if a 32-bit-timer is used it will overflow in around 111 seconds. Such a high frequency clock is not needed for the intended application of BSS. Using a prescaler would make overflows less frequent. For example a prescaler of 32 in combination with a 32-bit-timer will result in a frequency of 1.2 MHz and will overflow around every hour. The tick speed is $0.833\mu s$. With the goal of achieving a synchronization with an accuracy of at least 500 μs the period of the timer ticks should be at least less than 500 μs.

4.7 Timeline of the Protocol

In BLE there is a scanner and an advertiser. The advertiser send advertising packets using BLE. In the currrent setup the scanner is the client, and the advertiser is the server. The scanner sorts through advertising packets and initializes a connection with the correct wireless device. Which generates the connected event on both devices. At this connected event the value of both timers in the devices is noted. After the client has allowed indications the server sends the timer value it had at the connection interval. After receiving this value the client adjusts its own internal timer value, by using the elapsed time until the packet has been received and processed, as well as the timer value of the server. This removes the initial offset. After this indication has been received the client enables notifications. After notifications have been enables the startup phase begins where the server sends multiple notification packets to the client. Which the client uses to estimate its timer drift compared to the timer of the server. It should be noted that while the timer of the server also has a drift compared to the 38.4 MHz. The internal crystal does not oscillate

at exactly this frequency.This should not matter for the drift estimation, as the client timer is not corrected to be as close as possible to the 38.4 MHz frequency, but it is corrected to be as close as possible to the server timer. After the start-up phase the client will periodically adjust its internal timer, either skipping ticks, or recounting certain ticks. A known issue with this is that it is possible for a planned interrupt to thus be less accurate. If an interrupt has been set at counter value 1001, and the drift correction adjusts the internal timer on tick 1000 and increases the counter value by 10. It will mean that the interrupt is called 9 ticks later than intended. Since the interrupt is called when the counter value is either equal to or larger than compare value, the interrupt is still called and no logic is needed to adjust the interrupts. The best way to minimize this issue is by having the timer adjust period be short. For example in one second the expected drift is 10 μs. Thus if the timer is adjusted every second the maximum expected delay of an interrupt being handled is close to 10 μs. Using similar logic to what is used in the start-up phase, the drift correction and offset of the timer are adjusted periodically using notification packets, or the size of a connection interval. This should keep the timer in sync indefinitely.

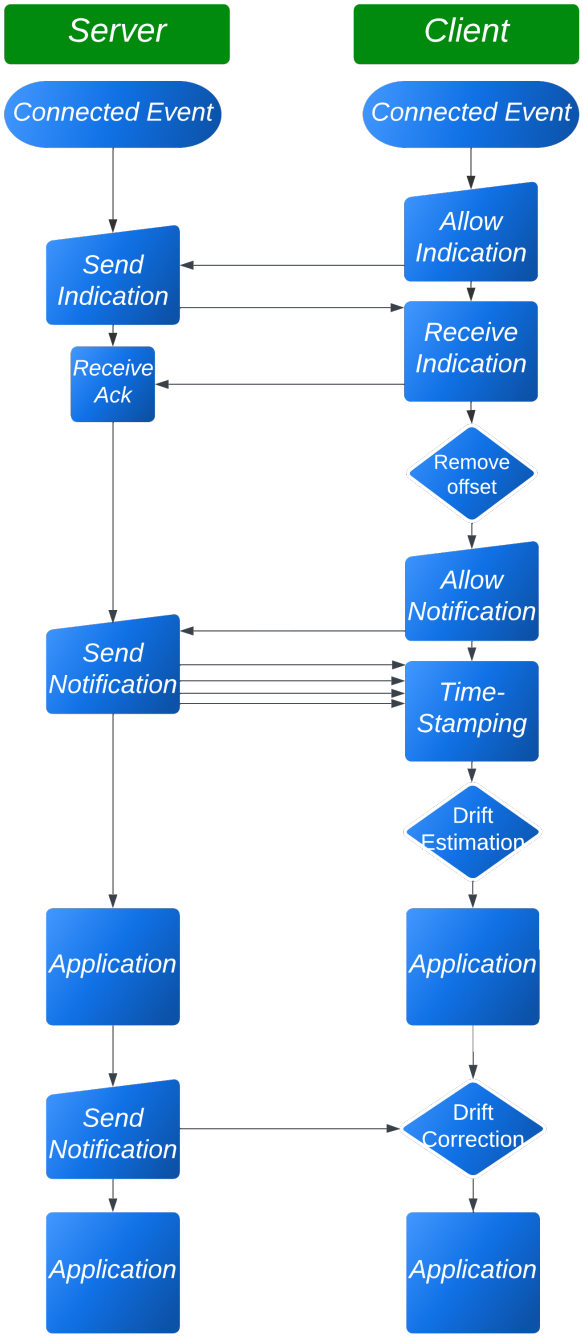

Figure 5: Overview of the entire timeline of the protocol. Starting with the connected event. To the Connection Phase. To the Drift Correction Estimation. To the Resync phase.

5 Experiments

5.1 Connection event

In the BLE-stack there is a connection event called on both the peripheral and the central device. Literature[4] suggests that this event can be used to synchronize the timers. When the connection event is called a pin on the board is set high. The moment that the pin was set high on the server, relative to when the pin was set high on the client, is measured. This provides the delay for both devices as to when this event is called. The timing of when a pin is set high is measured by an oscilloscope.

5.1.1 2 different hardware tested for the Connection event

The Experiment is done on both a SILABS explorer kit as well as a NORDIC development kit.

5.2 Accuracy and drift of the timers

Overtime the internal timers will drift away from each other. This experiment will describe and characterize this drift. For this experiment a 1 on 1 connection is setup between a server and client device. Every second a pin is set HIGH on both boards. These pins are measured by an oscilloscope and the delay between when the pins were set high are measured. Due to the connected event offset correction, for the first seconds of operation the devices will have no significant timing error. Over the duration of the experiment, due to the drift of the high-frequency crystal the moments that the pin is set high will drift. Through an oscilloscope this drift is measured. With typical ppm of +-5 the expected drift per second per device is in the range of 3-10 μs per second. To measure the drift over time a measurement of 1 hour is made.

5.3 Drift Correction

A startup protocol is implemented with the goal to reduce the drift of the timers over time. This is attempted through 2 estimation methods. Using fluctuations of the connection interval. And sending the server timer value using a notification. The first 10 seconds after the devices are connected were used to gather data for the drift estimation. This estimation is then used to adjust the internal timers every second, skipping or recounting skips. The client performs this estimation method using an average error algorithm, with the goal to follow the timer of the server. The error is defined as the difference between the expected length of the connection interval, compared to the measured connection interval on the client device. A pin is set high every second which is measured by an oscilloscope. An 1 hour measurement is performed which will measure the drift of the timers over time.

5.4 Connection interval Behaviour

The precision of a connection interval. This is checked by sending a notification, the shortest packet, each connection interval and checking internally and externally when the packet is send. There should be one connection interval delay between each notification packet that is send. Furthermore, the length and precision of a connection interval is tested. This is done in 2 ways, a pin is set high at the start of each connection interval, which is determined by when a notification is received. As well as having the internal timer of the client measure each time a notification is received internally, and comparing that to at what time it was expected that the time was received. This is a measurement that ran for 10 seconds and

5.5 Notification Behaviour

After performing the drift estimation using notifications it become evident that there was a behaviour not accounted for. The devices were becoming out of sync faster than if not drift estimation protocol was implemented. To investigate the cause of this the behaviour of notifications are analyzed. In this measurement every 100 ms a notification is containing the root timer value is send to the client device. This notification is send to the BLE stack based on a 100 ms period determined by the apptimer of Simplicity studios. This is due to the aforementioned fact that the high accurate timers are incompatible with the BLE stack. A less accurate timer is used. These measurements are performed with internal software timestamps. Due to the short timescale of the measurement, the major characteristics are still accurately measured. The measurement ran over a period of 10 seconds.

Bluetooth Timers Due to the before mentioned issue of not knowing when the notification interval starts, in combination with not being able to use the accurate timer interrupt to send a notification, results in an error. This is because the apptimer which is used to have a callback event to send a notification every 100 ms actually has a callback event every 99.988 ms. Due to the fact that the connection interval and the callback event have a different period, this will result in the server time that is send to the client to be each connection interval phase shifted by 11.6 μs. When comparing the internal timer of the client to the send timer value of the server this must be taken account, or it will over correct and thus get out of sync faster.

5.5.1 Notification Transmission time

The time it takes to send one notification with 32 bits of information. This is tested by setting the a pin high on the server when a notification is send, and setting a pin high on the client when the notification is received. And measuring the delay.

6 Results

6.1 Connected event

6.1.1 Silabs Explorer kit

From the experiment it became evident that the connection event called on both boards has a consistent time delay. On the server device the event is called on average 89 μs before the event is called on the peripheral device. With a standard deviation of 12 μs around this point.

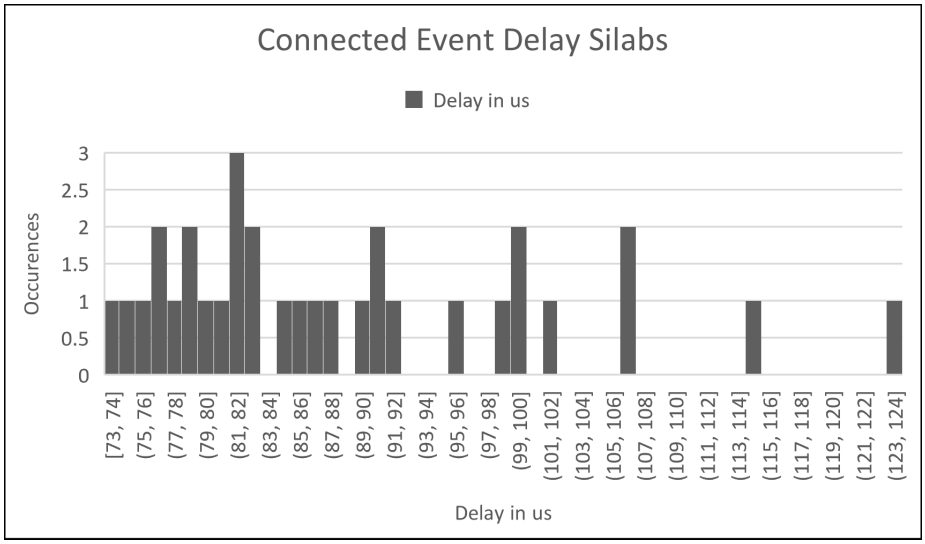

Figure 6: The perceived time delay of the connected event between central and peripheral Silabs explorer kit devices. Where the central device has the event before the peripheral device.

6.1.2 Nordic Development board

For the Nordic Development board the connection event is also called first on the server device before the peripheral device. This delay is either 450 μs or 520 μs. With very minor spread around these points.

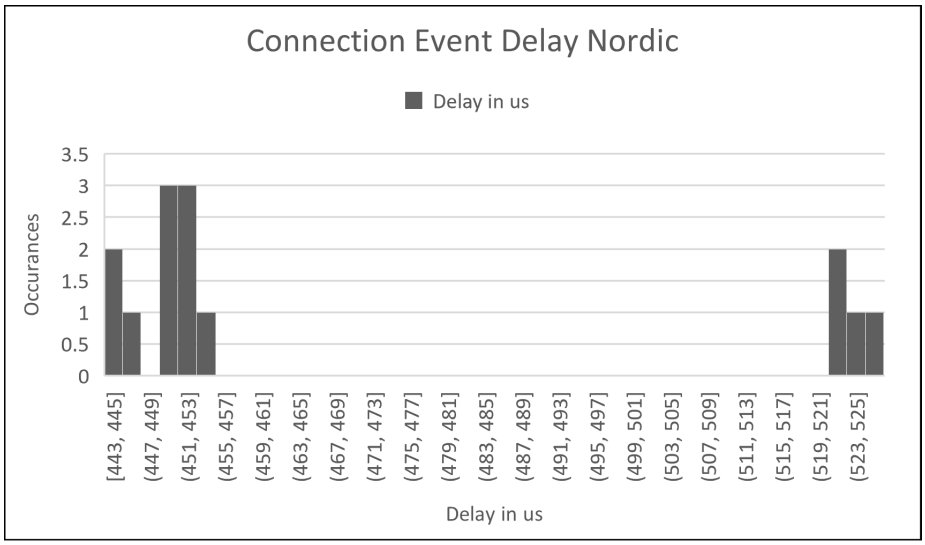

Figure 7: The perceived time delay of the connected event between central and peripheral Nordic Developer kit devices. Where the central device has the event before the peripheral device.

6.1.3 Behaviour and drift with only the connected event

For the next experiments only the Silabs explorer kit is used. What becomes evident from this measurement is that while the timers all have the same offset. Over the duration of the experiment these timers experience drift. This drift is in the region of 3-10 microseconds per second. This is in the typical range of high-frequency crystal. Below the graph which describes the timing error over time is displayed, with the X-axis being the central device. To reduce the starting offset the results from the previous experiment were used, which lead to adding a static delay of 90 μs to the timer of the client.

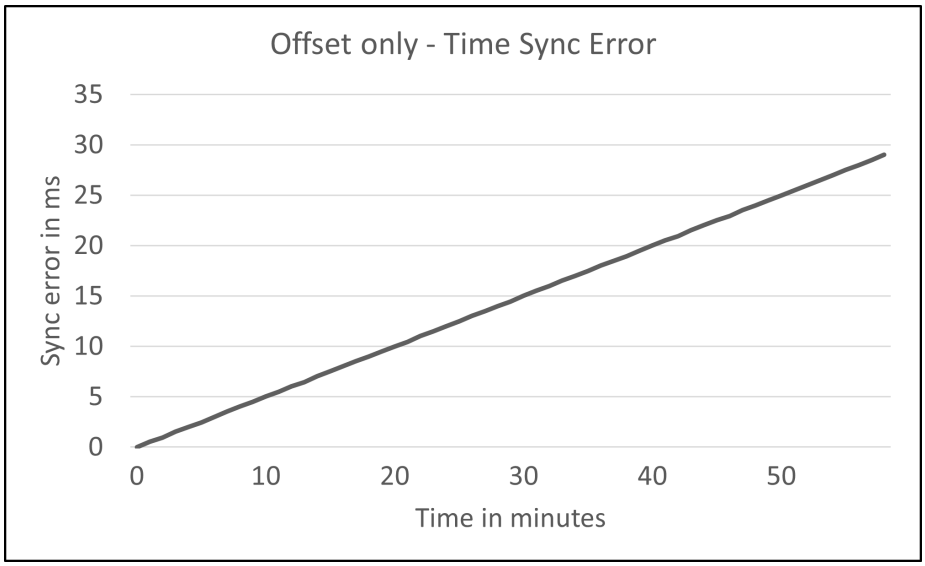

Figure 8: The sync error observed between the client and server device through a 1 hour measurement where the offset of both timers was adjusted.

6.2 Startup Drift correction - Connection Interval

Using both the connect event to correct the offset, as well as the startup drift estimation algorithm based on the connection interval fluctuation drift estimation. The following result is derived. What should be noted is that the total drift in Fig: 9 is less than the drift observed in Fig: 8.

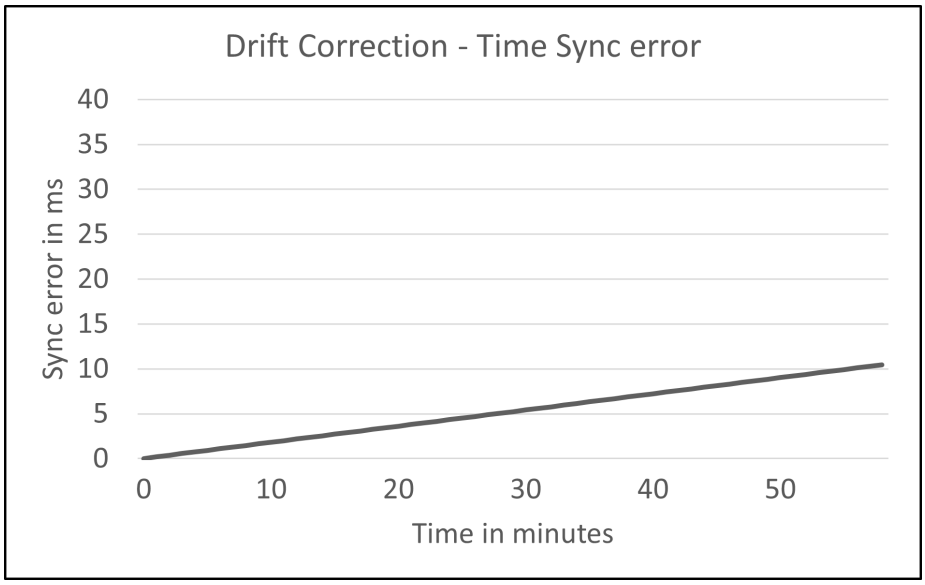

Figure 9: The sync error observed between the client and server device through a 1 hour measurement where the offset of both timers was adjusted and a startup protocol was implemented.

For the algorithm to work a large sample of measurements has to be taken. The Fig: 10 shows the behaviour over time. The connection interval measured has many instances of the interval stretching or shrinking, with the total net deviation over time being $+5\ \mu s$ at the end of the operation. This is less than the measurement of a single connection interval fluctuation, which is in the range of $+10\ \mu s$ and -20 μs.

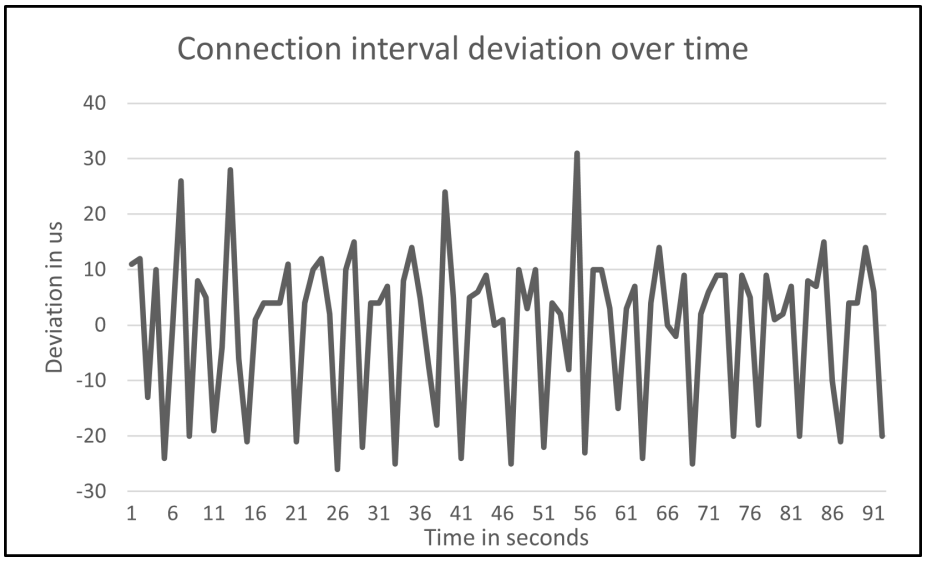

Figure 10: The microsecond stretching and shrinking of a connection interval as seen from the perspective of the client.

6.3 Drift correction - Notification

Sending the notification of the server time to the client and reading that value in order to sync the clock resulting in a major time drift. The timedrift was 100 μs per second. This is due to the apptimer that was used. The next experiment goes into more detail.

Notifications phase shift over time The apptimer callback has an accuracy up to one millisecond. The apptimer is what determines when a notification request is send to the buffer. Due to the low accuracy the moment that the notification is send to the buffer is shifting over time. In this graph it is possible to see the phase shift of when the notification was send to the buffer of the BLE-stack inside of a connection interval. The notification is send to the buffer in a different period than that the buffer is released. With an average of phase shift of 95 μs per second. The whole behaviour can be seen in Fig: 11. It is the time between the notification being send to the buffer and the notification being processed on the client side. The longest delay is around 100 milliseconds and 969 μs. And the shortest delay is 986 μs. To make the behaviour more clear 2 additional graphs have been provided which describe the jump to the beginning of the connection interval in Fig: 12. And the phase shift behaviour in Fig: 13 makes the walking of the notification through the connection interval more easily observable.

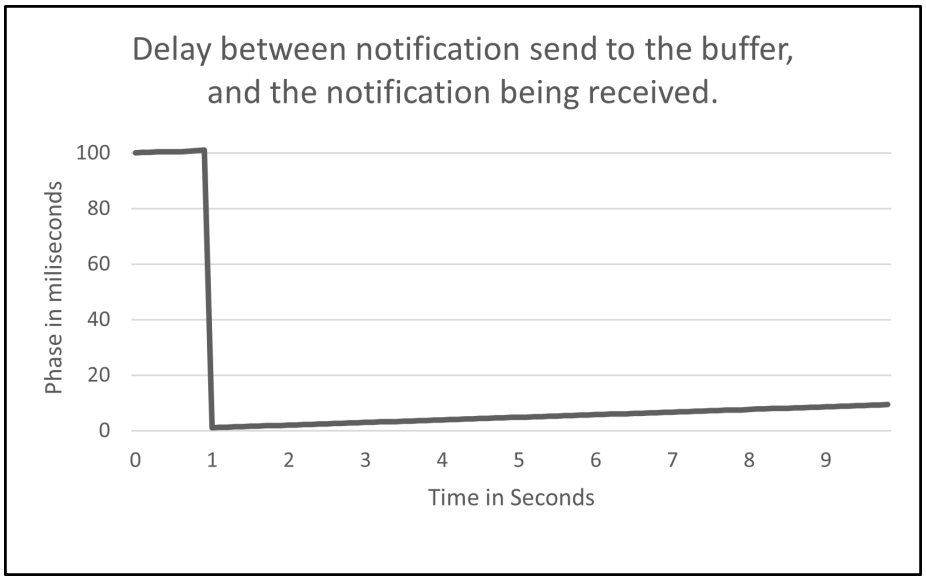

Figure 11: The delay of when the notification is send to the buffer of the Gatt server compared to when it is processed on the Gatt client.

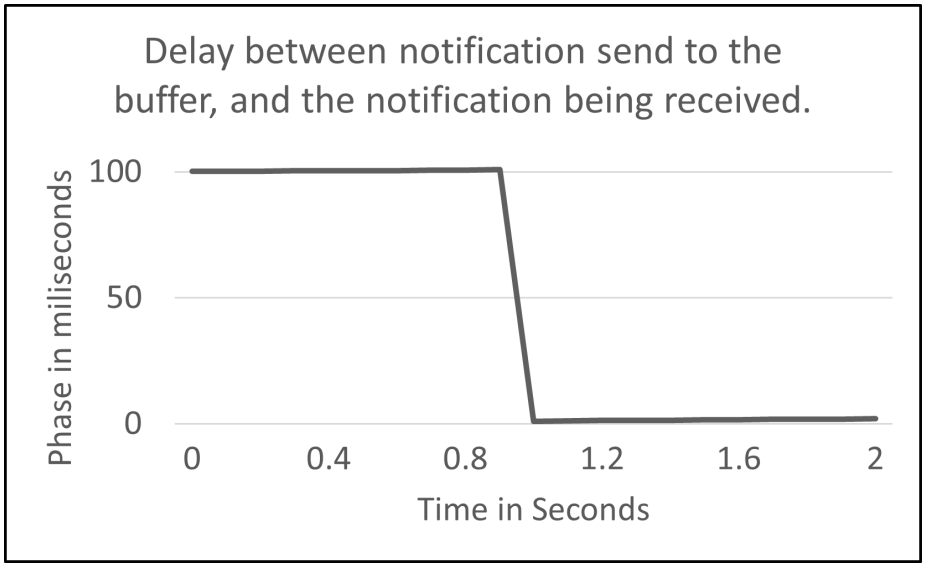

Figure 12: The delay of when the notification is send to the buffer of the Gatt server compared when it is processed on the Gatt client, zoomed in to accentuate the jump of the notification being at the end of the connection interval to being at the beginning of the connection interval.

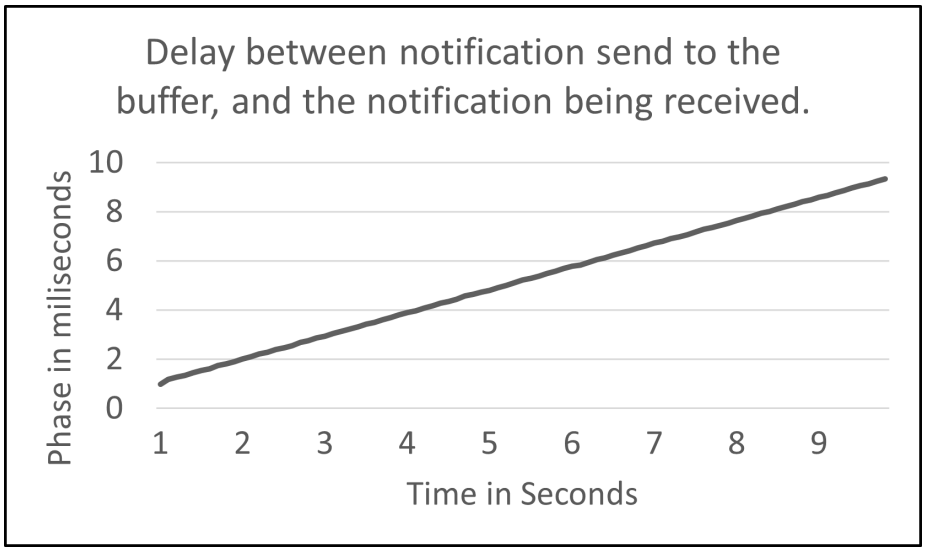

Figure 13: The delay of when the notification is send to the buffer of the Gatt server compared to when it is processed on the Gatt client. With the first second which contained the jump removed to make the shift over time more clear.

7 Discussion

7.1 Known issues

Due to the fact that the apptimer is not highly accurate, having a shorter period that what is set, it will result in the shifting of the notification is send to the buffer in comparison to the period of the connection interval. Due to the period of the apptimer callback being shorter than the period of the connection interval. Or an integer of connection intervals. It will eventually result in the timer value being send either twice in the same connection interval, in the case of the apptimer callback period being less than one connection interval, or the connection interval in which the notification is send will change throughout the operation. If every 10 connection intervals a notification is expected, this will result in the notification being received [0,10,20,30,40,50]. This consistency is needed for an accurate drift correction. However, as the period of the apptimer is slightly shorter than the period of the connection interval there are moment where this takes place: [0,10,19,29,39,49]. The occurrence of this issue is rare, but should be accounted for with additional logic.

The resync phase was not tested, this is due to the fact that the startup phase results were unsatisfactory. The resync phase is an extension of the drift estimation that is implemented during the startup phase.

7.2 Doubts about BLUESYNC sync error calculations

The BLUESYNC protocol described in [3] states a much lesser sync error over time than what would be expected and what was measured in this report. BLUESYNC in fig. 6 describes the sync error of the NRF51 and NRF52 boards, over a time period of 10 minutes. The SYNC error measured after 10 minutes was around 450 μs for the NRF51 and around 250 μs for the NRF52 if only the offset was removed. The most accurate crystal in the NRF51 has a ppm of 30. [8]. And for the most accurate crystal of the NRF52 has a typical ppm of 8. [5]. By only removing the offset the expected sync error after 10 minutes would be 0.6 millisecond per 1 ppm. With 8 ppm the expected typical sync error would be 4.8 ms after 10 minutes. The claimed value is 250 μs. This is almost a factor 20 difference between the documentation and the claimed values. Compared to the results of experiments in this paper, where the typical ppm of the crystal in the Silabs Explorer kit was 5 ppm. After 10 minutes of operation the error in drift is around 5 ms.

7.3 Problems with IDE

There were major setbacks due to problems experienced when using the IDEs Visual Studios, and Simplicity Studios. A problem observed in Simplicity studios is that it was for me impossible to get the central device to the be the server. The peripheral device had to be the server for the functions to work as intended. The first intended structure was to have a central server surrounded by clients, and that these clients would listen to notification packets from the server to synchronize their timers. This structure had to change, and each peripheral node would become their own server. Notifications can only be send from the server to the client in Simplicity studio. And due to the speed of notifications packets, they are most commonly used to synchronize timers, for example in [3]. The documentation on this matter for Simplicity studio is limited, and there is no indication that the server role can only be fulfilled by an advertiser device. These problems lead to the tests being performed one to one.

7.4 Connected-Event

The connected event can be used to remove the offset between timers effectively and consistently. The variation of delays can be explained by random processes or error in the system. For the Silabs explorer kit the system the connected event sync error between the client and server was on average 89 μs. But more important than the average delay, the spread was very small which made using the connected event to remove the offset an effective method. With a standard deviation of 12.4 μs around this point, ensured that the original offset is eliminated as much as possible. On the Nordic boards the behaviour was centered around 2 points. That being 450 μs and 520 μs. Where the 450 μs delay occurred more often than the 520 μs.s When adding the offset value to the central device to ensure that the offset are minimized at the start of the

operation it would be better to pick the value in the middle of these 2 values. This would give an offset of 485 μs. While this is not the best method to reduce the offset the most on average. It does provide more consistency. If the average error was the most important point then there would be cases where the actual sync error would be as great as 70 μs. And at other times the sync error would be 0 μs. This is a large spread, and to create a protocol that is able to have a synchronized timebase between devices and is scalable, it would be better to reduce the spread. Therefore for the Nordic Development kits the recommendation is to have account for a delay of 485 μs.

7.5 Offset-only Drift

The drift that was expected by only correcting the offset was about the 5 ppm per device. In one hour a drift of about 29 ms was measured. This is too great for time synchronization and this is proof that another different method should also be implemented, namely the drift estimation. If this result is calculated to an ppm error between the different timers it is is an error of

$$PPM = \frac{TimeError}{Measurementtime} \tag{1}$$

$$PPM = \frac{29ms}{3600s} \tag{2}$$

Results in an ppm of 8 ppm. Considering both timers have a typical ppm error of +- 5 ppm, this result is within expected bounds.

7.6 Startup-Drift

A startup-drift correction based on the fluctuations of connection intervals was used to estimate the drift of the client device compared to the server. After the implementation it resulted in a smaller synchronization error after the one hour measurement when compared to no drift correction method. While this result is promising, When looking at the behaviour of the connection interval fluctuations it becomes evident that the connection interval method does not provide the relevant information needed for an accurate drift estimation. In order to observe whether the connection interval had a tendency to be shorter, or longer on average, the length of 100 connection intervals was measured. What became evident from this measurement is that the average fluctuation approached 0. With the oscilloscope a synchronization error was measured over time. If it was possible to measure the synchronization error with the connection interval, then instead of the average approaching zero, it would approach a constant. This is not the case, therefore using the connection interval fluctuations for to estimate the drift is not possible, and it was a coincidence that the synchronization error over time improved, this could be due to the fact that different boards, or a different environment was used. Another method that should be tested if the for instead the scanner device, in this setup the client device, that the

advertising device, in this setup the server, measures the fluctuations of the connection interval. It is a possibility in that with this setup the fluctuations could be used to estimate the drift.

7.7 Notification packets

When using notification packets to read the server time it is important to know all the delays that are involved. Some of these delays are due to transmission and processing. From the experiments it became evident that for the smallest packet necessary to send the whole 32 bits of the timer value, this transmission and processing time was less than 1 ms. As becomes evident from the shortest delay in Fig: 11. The second delay that should be accounted for is when the notification is send to the buffer from the server side. There is an issue where it is unknown when the connection interval of a device starts, and when this connection interval ends. This not knowing when a packet will be send in the application layer means the processing of when it is send will be handled by the BLE stack. Using the connected event and the typical drift, it is know that for the first second of operation the devices will be in sync. In this brief period notifications from the server should be send to the client, and the client should take account of what the server time is written in the packet, and what the time it was when the packet was received. Using the assumption that at the start of the operation the timers are in sync, and that notification transmission and processing takes the same time for every packet, it will be possible to write logic to observe the phase shift of the server time through the connection interval. And then when reading the server time from the packet, this estimated phase shift is taken into account to correct the drift. This was not developed in this project and could be a part of a next research project. Without this being developed it was not possible to test an indefinite synchronization protocol.

7.8 Connected Event Expansion routes

There is a possibility of using the connected event for the drift estimation, or by reconnected the devices periodically that time synchronization is achieved. The connected phase provides a very small synchronization error between 2 devices. When devices lose connection, they do not lose data. Therefore it is possible to use multiple connected events to estimate the drift of the device. This would need the devices to disconnected and reconnect again. It is also possible to have the devices disconnect during the operation of a application, and have them reconnect during the operation. This can then be used to synchronize the timers, achieving time synchronization.

7.9 Next steps

For potential next steps in the build of an indefinite synchronization protocol would be to continue work on the notification logic. The phase shift in the connection interval should be calculable. This gives a phase shift estimation.

This phase shift estimation can be used to then create a drift estimation, based on the value of the server timer that is send. A possible solution for this is discovering a method to use a highly accurate timer to provide the interrupt that send the notification to the BLE stack. Another potential route is to change the setup, where instead of the client timer being matched to the server timer, that the server timer is matched to the client timer. For this there is a possibility that the drift estimation based on connection intervals can be used, and should be investigated. Or by keeping the setup, but instead of having a single central server, that there are multiple peripheral servers which are connected to a central client. And that these peripheral server devices will sync their timer with the time of the central client. This could potentially be done with the write without response function in BLE.

8 Conclusion

The objective of this project was to investigate the feasibility of developing a Bluetooth Low Energy (BLE) synchronization protocol for the BSS application. Through experiments and a literature review, an effective approach for eliminating the initial timer offset was devised and validated. Additionally, a theoretical framework for achieving indefinite device synchronization was proposed.

An in-depth exploration was conducted on utilizing the fluctuation of connection intervals to estimate the drift between two timers. However, the obtained results were discouraging, indicating that this approach may not be viable for achieving accurate synchronization. As an alternative, the utilization of notifications to transmit timer values and leveraging this method for synchronization emerged as a promising path, offering potential for future applications.

References

[1] A. Akay, *Evoked Potentials*. InTech, 2012.

[2] T. Zhang, J. Lu, F. Hu, and Q. Hao, "Bluetooth low energy for wearable sensor-based healthcare systems," in *2014 IEEE Healthcare Innovation Conference (HIC)*, 2014, pp. 251–254.

[3] F. Asgarian and K. Najafi, "Bluesync: Time synchronization in bluetooth low energy with energy-efficient calculations," *IEEE Internet of Things Journal*, vol. 9, no. 11, pp. 8633–8645, 2022.

[4] A. Bideaux, B. Zimmermann, S. Hey, and W. Stork, "Synchronization in wireless biomedical-sensor networks with bluetooth low energy," *Current Directions in Biomedical Engineering*, vol. 1, no. 1, pp. 73–76, 2015. [Online]. Available: https://doi.org/10.1515/cdbme-2015-0019

[5] Nordic Semiconductor ASA, *nRF52832 Product Specification*, 2019. [Online]. Available: https://infocenter.nordicsemi.com/pdf/nRF52832$_PS_v$1.1.*pdf*

[6] S. Labs, *BGM220P Wireless Gecko Bluetooth Module Data Sheet*, 2020. [Online]. Available: https://www.silabs.com/documents/public/data-sheets/bgm220p-datasheet.pdf

[7] Silicon Labs, "Bluetooth connection flowcharts," Online, 2023, accessed: July 10, 2023. [Online]. Available: https://docs.silabs.com/bluetooth/3.2/general/connections/bluetooth-connection-flowcharts

[8] Nordic Semiconductor ASA, *nRF51 Series Reference Manual*, 2014. [Online]. Available: https://infocenter.nordicsemi.com/pdf/nRF51$_RM_v$3.0.*pdf*