CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Remote Operation and Connectivity for ReLog

Master's Thesis in Embedded Electronic System Design

ARMAN VASEI

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Remote Operation and Connectivity for ReLog

ARMAN VASEI

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Remote Operation and Connectivity for ReLog

ARMAN VASEI

Supervisor: Per Larsson-Edefors, Department of Computer Science and Engineering
Advisor: Albin Anner, ReVibe Energy
Examiner: Lena Petersson, Department of Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

Remote Operation and Connectivity for ReLog

ARMAN VASEI
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Vibration analysis is a process that monitors the magnitude and behavior of vibration signals within a component or structure, to detect potential harmful vibration events and observe general condition of the system under test. ReVibe Energy has developed the ReLog vibration data logger, a measurement tool used to record vibration and temperature with very high precision. Few customers asked for time synchronization feature on loggers since they need to keep track of time of important events in different parts of a device. In this thesis we investigated possibility of implementing time synchronization protocols on ReLog. We implemented a few well-known protocols and achieved timing accuracy of 10 ms. We also implemented a number of remote operation features for ReLog which makes it more simple to use for remote applications in dangerous and inaccessible environments.

Keywords: Vibration Sensing, Time Synchronization, Data Streaming, Wireless.

Acknowledgements

I would like to express my special thanks of gratitude to Erik Godtman Kling and Victor Börjesson who trusted me and gave me the opportunity to be part of the ReVibe Energy. Secondly I'm extremely grateful to thank my superviors at ReVibe, Albin Anner and Felix Eriksson, who guided and supported me through the project. I would like to extend my sincere thanks to my academic supervisor, Per Larsson-Edefors, who drew an outline for the thesis scope and gave precious comments on the report. Finally I would like to thank my examiner, Lena Peterson, for her comments and feedback on the report.

Arman Vasei, Gothenburg, June 2021

Contents

Abbrevations **xi**

List of Figures **xii**

List of Tables **xiii**

1 Introduction **1**
1.1 Objectives . . . 2
1.2 Previous Works . . . 2
1.3 Limitations . . . 3
1.4 Thesis Outline . . . 3

2 Theory **4**
2.1 ReLog Data Logger . . . 4
2.2 Clock Behaviour . . . 5
2.3 Network Layers and Protocols . . . 5
2.4 Time Synchronization Protocols . . . 6
2.4.1 Network Time Protocol . . . 7
2.4.2 Precision Time Protocol . . . 8
2.4.3 Reference Broadcast Synchronization . . . 9
2.4.4 Other Time Synchronization Protocols . . . 9
2.5 Data Streaming . . . 10
2.5.1 Transport Layer Protocols . . . 10
2.5.2 Message Queuing Telemetry Transport . . . 10

3 Methodology and Tools **12**
3.1 Hardware . . . 12
3.1.1 SAM D21 Xplained Pro . . . 12
3.1.2 ATWINC3400-XPRO . . . 13
3.1.3 NUCLEO-L4P5ZG . . . 13
3.2 Software Tools . . . 14
3.2.1 Microchip Studio . . . 14
3.2.2 STM32CubeIDE . . . 14
3.3 Implementation . . . 14
3.3.1 Time Synchronization System . . . 15
3.3.2 Data Logger . . . 17
3.4 Triggers . . . 17

4 Results **18**
4.1 Clock Drift . . . 18
4.2 Time Synchronization . . . 19
4.3 Comparison of Protocols . . . 23
4.4 Time Synchronization for Data Logger . . . 24
4.5 Data Streaming . . . 25
4.6 Triggers . . . 25

5 Discussion and Conclusion **27**
5.1 Discussion . . . 27
5.1.1 Transport Layer Protocol . . . 27
5.1.2 PTP Experiment Results . . . 27
5.1.3 Offset . . . 27
5.1.4 Data Streaming Rate . . . 28
5.1.5 Integration into the Main Product . . . 28
5.1.6 Triggers . . . 29
5.2 Conclusion . . . 29

Bibliography **30**

Abbreviations

ADC	Analog-to-Digital Converter
API	Application Programming Interface
BLE	Bluetooth Low Energy
FTSP	Flooding Time Synchronization Protocol
HAL	Hardware Abstraction Layer
MQTT	Message Queuing Telemetry Transport
NTP	Network Time Protocol
PPM	Parts Per Million
PTP	Precision Time Protocol
RBS	Reference Broadcast Synchronization
RTC	Real-Time Clock
SDIO	Secure Digital Input-Output
SNTP	Simple Network Time Protocol
SPI	Serial Peripheral Interface
TC	Timer/Counter
TCP	Transmission Control Protocol
TPSN	Time-sync Protocol for Sensor Networks
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
WSN	Wireless Sensor Network

List of Figures

2.1 ReLog data logger 4
2.2 IEEE 802.11 WLAN relation to the network layers 6
2.3 NTP stratum levels 7
2.4 NTP time synchronization 8
2.5 PTP time synchronization 8

3.1 SAM D21 Xplained Pro evaluation kit 12
3.2 ATWINC3400-XPRO 13
3.3 NUCLEO-L4P5ZG development board 14
3.4 System block diagram 15
3.5 Experimental system setup 15

4.1 Clock drift and offset between server and client 18
4.2 Offset between server and client after applying SNTP 19
4.3 Distribution of offset between server and client after applying SNTP . 20
4.4 Offset between server and client after applying PTP 21
4.5 Distribution of offset between server and client after applying PTP . 21
4.6 Offset between server and client after applying RBS 22
4.7 Distribution of offset between server and client after applying RBS . . 22
4.8 Synchronized square waves with SNTP 23
4.9 Captured sine waves from two synchronized STM32 Nucleo boards . . 24
4.10 Cross correlation between captured signals 25
4.11 Vibration log without applied shock 26
4.12 Vibration log with applied shock 26

List of Tables

2.1 Comparison between time synchronization protocols 10

4.1 Comparison between time synchronization protocols 23

1
Introduction

The wireless sensor network (WSN) is one of the most important parts of internet of things (IoT) systems. Sensors can communicate with various standards such as WiFi, LTE, or Bluetooth low energy (BLE). One of the critical aspects within WSN is time synchronization [1], which is important for many sensor network applications that require very precise mapping of gathered sensor data with the time of the events.

Another important part of IoT systems are sensors. Sensors can measure various information from temperature and humidity to position and motion [2]. One significant application of sensors is vibration measurement. Vibration is movement or oscillation of a device or component. The object can vibrate in two modes: free vibration and forced vibration. Free vibrations are oscillations for which the total energy and amplitude of vibration stays the same over time. Natural frequency refers to the frequency at which a structure is going to oscillate after an impact. Forced vibrations occur when the object is forced to vibrate at a specific frequency by a periodic force. Forced vibration at or close to the natural frequency causes resonance which means that over time the vibration can become quite large in amplitude. If a structure has natural frequencies that match normal environmental vibration, then the structure vibrates more violently and probably collapses. That explains why vibration measurement has great importance.

ReVibe has developed the "ReLog" vibration data logger [3], a measurement tool which is able to record vibration and temperature with very high precision. The ReLog data logger can sample data with rates between 125 Hz and 32 kHz. ReVibe has recently been approached by a company in the oil industry. The company asked if it is possible to implement time synchronization among several units of ReLog. The reason for this request was that the company was going to install several units of ReLog on different sides of a ship engine and it need to keep track of every interesting event on all sides and compare the effect of each event on all sides, so the loggers should capture data while they are synchronized in time. In the meeting with ReVibe customer specified following requirements:

- Sample rates higher than 1 kHz will be seldomly used.
- Vibrations in the range of 0.3 -100 Hz are important.

So based on the given specification, a wireless communication between ReLogs should be established and a time synchronization protocol has to be implemented which must provide time accuracy of 1-5 ms.

Several other customers asked ReVibe if it is possible to add some remote operation features on ReLog. Some of the ReLog use cases are logging vibration inside mines or engines. Since these environments are not easily accessible or sometimes even they are dangerous, it is wiser to access data remotely. One merit of using WiFi transmission is that it can be used for data streaming. Hence a WiFi connection to another device or a cloud server could be a good solution. In general remote operation is a necessary feature for data loggers. Another useful feature is ability to log data on specific time and date or periodically. For this purpose an automatic mechanism to start and stop data logging is required. Customers also asked for acceleration triggers. Currently ReLog can save up till 128 GB of data. But that may contain no important event. One of the solutions to solve this issue is to add a shock detection mechanism to the logger. In this case ReLog start logging data after receiving a significant shock.

1.1 Objectives

The main goals and objectives for this thesis are:

- Search and compare different wireless modules and choose the one with lower cost and simpler available drivers and application programming interface (API).
- Implement various time synchronization protocols for accurate synchronization of measurements and pick the best one in terms of timing accuracy.
- Combine the time synchronization system with ReLog to measure final achievable time accuracy.

Our secondary goal is to investigate if the chosen module is suitable for data streaming to another device or a cloud server. And our final goal is investigate if it is possible to add time and acceleration triggers to the ReLog.

1.2 Previous Works

Currently customers buy ReLog for vibration measurements in mines or shipping industry. Each ReLog will be installed in the desired location. After end of measurement, ReLogs will be collected and connected to a host PC. Vibration data of each ReLog can be plotted and investigated independently. Currently there is no support for synchronization of two or more units of ReLog. To solve this issue we have to find a suitable wireless transmission standard and implement a time synchronization protocol between ReLogs.

Several time synchronization protocols have been proposed (e.g., network time protocol (NTP), reference broadcast synchronization (RBS), timing-sync protocol for sensor networks (TPSN)) which are briefly described in [5]. These protocols differ in terms of implementation and accuracy. Hilmersson and Gummesson [6] investigated accuracy of existing protocols. They implemented the RBS protocol (method in which receivers use broadcasts to compare clocks) on a network consisting of NINA-B1[7] module, which only supports the BLE standard. They also investigated the

impact of synchronization interval, network load, and number of sensors on accuracy. Other synchronization protocols were not implemented in this work and other standards, such as WiFi, were not tested.

Sheng et al.[8] have implemented the RBS protocol over network of nRF24L01 by Nordic Semiconductor [9] (which is now out-dated and replaced by newer products) and achieved accuracy of 410 µs. But they have not mentioned which wireless transmission standard they have used.

The limitation of previous works was they just implemented one specific time synchronization protocol. They also did not give any details about their design decisions and methodology. Our main goal is to implement well-known protocols as much as possible to compare different protocols in terms of time accuracy. But similar to the previous works we are going to limit ourselves to just one wireless transmission standard which is WiFi.

1.3 Limitations

In this thesis we study the feasibility of implementing time synchronization for ReLogs with aid of evaluation kits. But we do not integrate our solution into the final product, since PCB design takes too long time and also the original firmware of ReLog would have to be modified. Also the current package of ReLog is made of metal which shields WiFi signals. So with current design, wireless module cannot be integrated into ReLog.

1.4 Thesis Outline

The rest of this report is organized as follows:

- Chapter 2 starts with brief description of ReLog. It also presents the theory behind time synchronization and describes well-known existing protocols.
- Chapter 3 introduces the hardware and software tools which we used in our experiments. It also includes our methodology, proposed solutions and design decisions.
- Chapter 4 contains the results of our experiments.
- Chapter 5 discusses interesting observations in results from chapter 4. It also mentions some unexpected results and possible explanations for them. And finally it ends with a conclusion.

2
Theory

In this chapter, the theory behind ReLog, clock drift, network layers, and also time synchronization protocols will be presented.

2.1 ReLog Data Logger

ReLog is a data logger which is able to measure vibration and temperature. It has two accerlerometers for measuring the vibration. The primary accelerometer supports 4, 8, 16, and 32 kHz sampling rate. The secondary accelerometer is able to sample data at rates 125, 250, 500, 1000, 2000, and 4000 Hz. Measurement of temperature is optional. All these settings can be configured in the root folder of ReLog before start of logging.

Figure 2.1: ReLog data logger

Each accelerometer samples data in three dimensions and each sample has two bytes. All samples will be temporarily stored in a buffer and then they will be written into an SD Card. Depending on size of the SD card (which can be 32, 64, or 128 GB) and sampling rate, ReLog can record data up till 155 hours. The interface between the accelerometers and ReLog processor is serial peripheral interface (SPI) and the

interface between processor and SD Card is secure digital input-output (SDIO).The data logged by ReLog will be saved internally into *.wav* format which can be read and plotted by *VibInspect* [4] software.

2.2 Clock Behaviour

The need for time synchronization comes from the behaviour of clocks in the computer systems. According to [5], different nodes in a network can represent different times due to three main reasons: The nodes have been started at different times, the quartz crystals of each node might be running at slightly different frequencies which cause divergence of the clock values from each other (clock skew), and the frequency of the clock changes variably over time because of environmental conditions such as temperature (clock drift).

In other words clock drift is an accumulated effect of a clock rate that differs from reference time. If *C(t)* represents the clock time, for a perfect clock we have:

$$\frac{dC}{dt} = 1 \tag{2.1}$$

If this derivative is greater or less than 1, it means clock is faster or slower than reference respectively. Also if the derivative is equal to 1 that does not mean the clocks are synchronized, since they might have offset due to different initialization times. The drift rate, which is measured as the offset between the clock and a precise reference clock, is often expressed as a ratio in *parts per million (PPM)*[10].

2.3 Network Layers and Protocols

One of the most well-known models for networks is *open systems interconnection (OSI)* [11] model which is depicted in Figure 2.2. This model partitions the flow of data in a communication system into five abstraction layers:

- **Physical Layer**: This layer handles the transmission and reception of raw bits over a physical medium.
- **Data Link Layer**: This layer defines the protocol to establish and terminate a connection between two physically connected devices and also defines the protocol for flow control between them.
- **Network Layer**: This layer provides the content of a message and the address of the destination node and letting the network find the way to deliver the message to the destination node.
- **Transport Layer**: This layer creates segments by dividing a long message into smaller messages and provides a reliable connection.
- **Application Layer**: This layer interacts with user and software applications that implement a communicating component. In many sources there are two more layers between Transport Layer and Application Layer: *Session Layer*

and *Presentation Layer*. But sometimes these layers are considered within Application Layer.

Several protocols and standards have been proposed to communicate between nodes in the network. Different standards have different physical layer and data link layer architecture but they do not affect higher layers. One of the well-known standards is called *IEEE 802.11 WLAN* [12] which is commonly known by name of WiFi. The 802.11 standard define frame types for transmission of data as well as management and control of wireless links. These frames are divided into three functions: management frames, control frames and data frames. Each frame consists of a media access control (MAC) header, payload and frame check sequence (FCS).

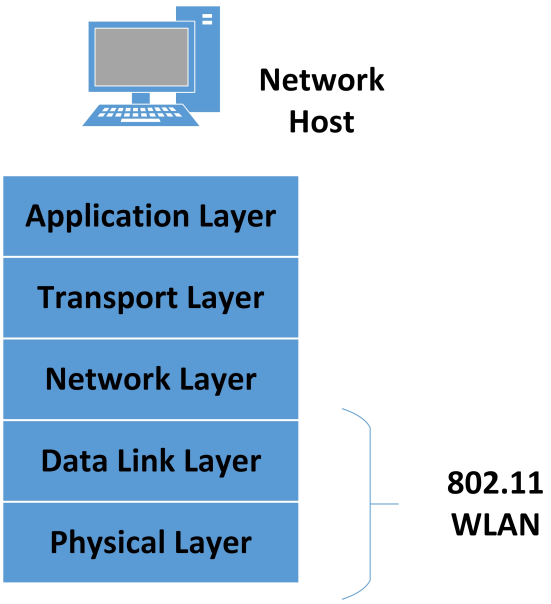

Figure 2.2: IEEE 802.11 WLAN relation to the network layers

The time at which an event happens is called *Timestamp*. Precision of timestamp can vary from a date to sub-nanoseconds[13]. Timestamping can be performed in any layer of network. *Hardware Based Timestamping* occurs in Physical Layer or Data Link Layer and it is the most precise way to capture time since it removes the delay between application and lower layers. But this method requires hardware support. *Driver Based Timestamping* is done within either Network or Transport Layer. This method also needs support from drivers. The least accurate method, which is *Application Based Timestamping*, captures the time from the running application. This method does not require any support from hardware or driver but in the other hand it is less accurate.

2.4 Time Synchronization Protocols

As we mentioned before, several time synchronization protocols exist. In this section we will introduce the most well-known protocols.

2.4.1 Network Time Protocol

The network time protocol (NTP) [14] is one of the oldest standards for time synchronization between devices in networks. In this protocol a number, called *Stratum*, is assigned to each device on the network. Devices in Stratum 0 (which are generally atomic or GPS clocks) have the most accurate clocks and act as time reference. But ordinary devices in networks cannot be connected to the Stratum 0 time references, so Stratum 0 devices are usually used as a reference clock and synchronisation source for a Stratum 1 time servers. NTP Stratum levels are illustrated in Figure 2.3. NTP can support up till 16 Stratum levels.

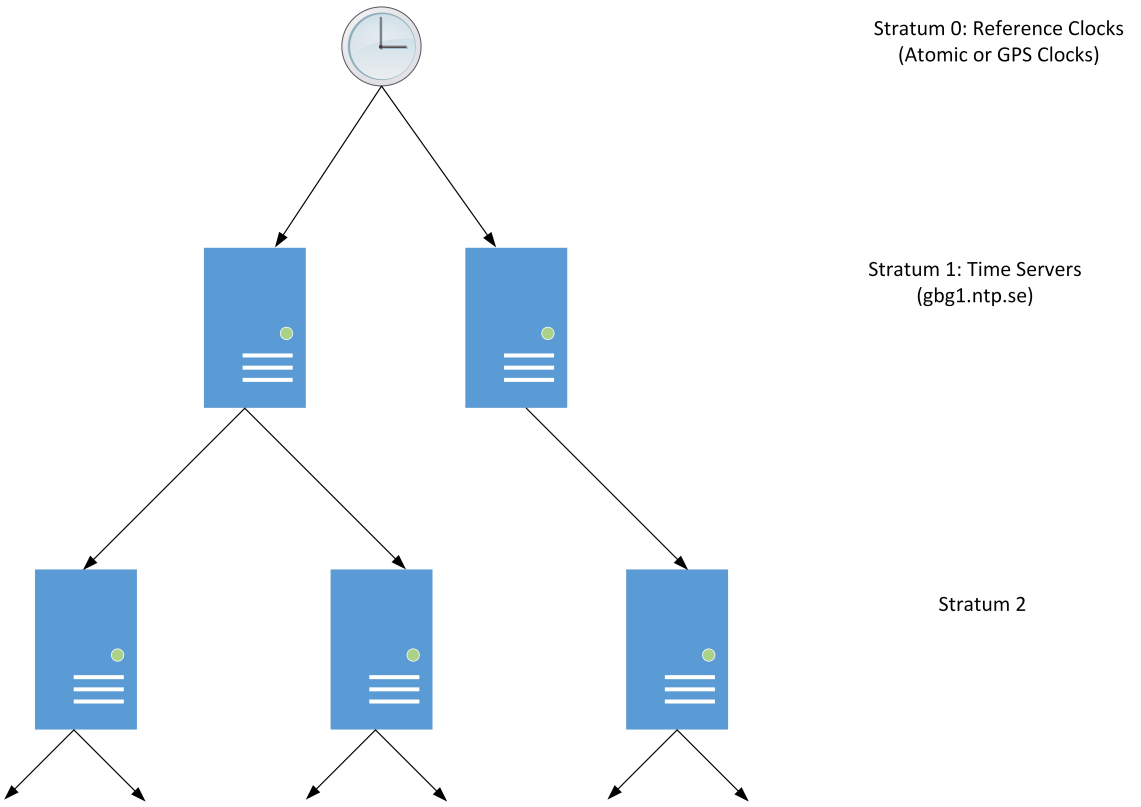

Figure 2.3: NTP stratum levels

In this protocol, the client sends a packet to server and timestamps the sending time which we call it t_0. The server timestamps the received packet at t_1 and sends a response packet at t_2. The client will receive the response packet at t_3. This two-way message is depicted in Figure 2.4. NTP assumes a symmetric propagation delay between server and client. The offset and propagation delay can be calculated as:

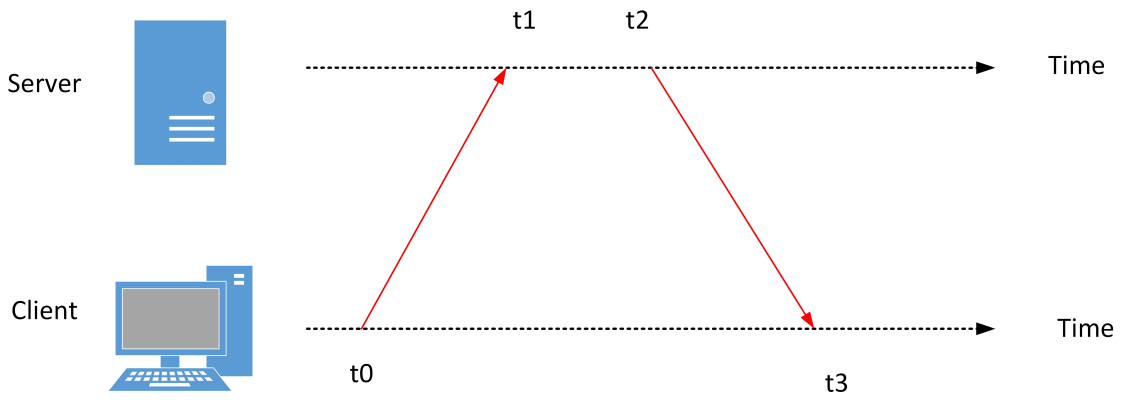

Figure 2.4: NTP time synchronization

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

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

After calculation of the offset, the client adjusts its own clock frequency to reduce the offset gradually. In NTP protocol values of offset and delay are saved for further statistical analysis. The simple network time protocol (SNTP) is a less complex version of NTP which does not store offset and delay values over time and just simply add or subtract the offset to the client's time. Since SNTP is simpler and more light-weight than NTP, it is a suitable option for embedded systems.

2.4.2 Precision Time Protocol

The precision time protocol (PTP) [15] is one of the most precise standards which was originally designed for wired networks. In this protocol the master sends a *Sync* message at time t_1 and slave captures it at t_2. Then the slave sends a *Delay_Request* message to the master at time t_3. the master captures the request at time t_4 and sends this timestamp in *Delay_Response* message. This procedure can be seen in Figure 2.5.

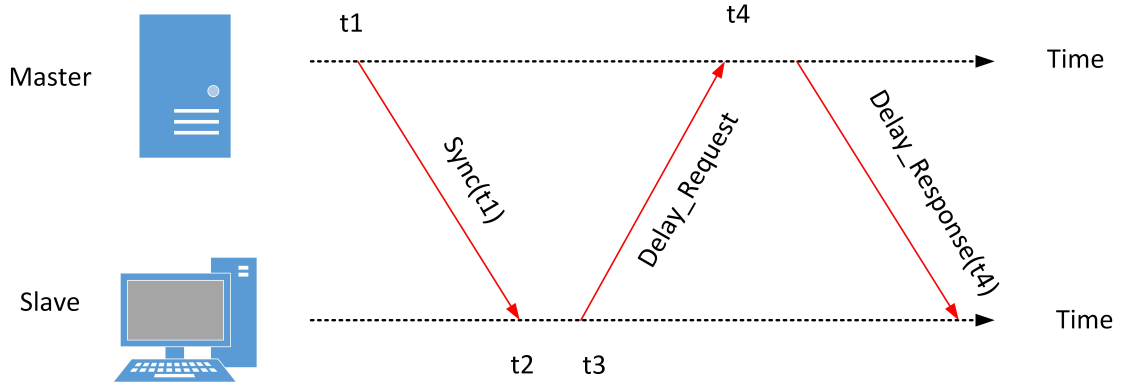

Figure 2.5: PTP time synchronization

PTP distinguishes between master-to-slave ($delay_{ms}$) and slave-to-master ($delay_{sm}$) propagation delays. The offset and delays can be calculated as follow:

$$\text{offset} = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{2.4}$$

$$\text{delay}_{ms} = t_2 - t_1 \tag{2.5}$$

$$\text{delay}_{sm} = t_4 - t_3 \tag{2.6}$$

Then the slave should adjust its clock to reduce the offset with master.

2.4.3 Reference Broadcast Synchronization

The previous mentioned protocols are used to synchronize receiver with transmitter. But the reference broadcast synchronization (RBS) [16] method synchronizes receivers among each other. One transmitter broadcasts n reference packets to all receivers. Receiver i timestamps packet k at time $t_{i,k}$. After transmission of all reference packets, receivers share information between each other. Node i can calculate its offset with node j by following equation:

$$\text{offset}(i, j) = \frac{1}{n} \sum_{k=1}^{n} (t_{j,k} - t_{i,k}) \tag{2.7}$$

One of the main benefits of RBS is good scalability within wireless sensor networks. Since this method does not need a point-to-point connection for synchronization. One central node broadcasts reference packets and every receiver can capture it. So the number of receivers can be increased as many as needed.

2.4.4 Other Time Synchronization Protocols

Beside the three mentioned protocols, several other methods exist. The time-sync protocol for sensor networks (TPSN) [17] is based on a hierarchical structure within network. One node acts as root and assign level 1 to the neighboring nodes. Nodes in level 1 perform similar procedure to their neighbors which leads to creation of spanning tree. This procedure is called *level discovery phase*. After this stage, each pair of nodes in two consecutive levels perform synchronization similar to the NTP.

Another protocol, called flooding time synchronization protocol (FTSP) exists which is similar to TPSN. The only difference is that the network structure is mesh topology instead of spanning tree. Also a few hybrid time synchronization approaches have been introduced as can be seen in [18]. A comparison between various protocols can be seen in Table 2.1:

Table 2.1: Comparison between time synchronization protocols

Protocol	Accuracy	Scalability
NTP	1 ms	Good
PTP	1 µs	Good
RBS	29.1 µs	Good
TPSN	16.9 µs	Poor
FTSP	1.48 µs	Average

We should note that accuracy values in Table 2.1 are average values and they might vary based on implementation. For example the 1 µs accuracy of PTP will be achieved by hardware-supported timestamping. If such a support does not exist, then the accuracy would be much lower.

NTP and PTP are quite easy to implement since all nodes just need to establish a point-to-point connection with a reference. RBS also is simple to implement since it is based on broadcasting. Simplicity of implementation makes all these three protocols scalable. TPSN and FTSP create a topology within the network which makes them more complex rather than other protocols. This feature reduce the scalability of these protocols. Generally TPSN suffers from poor scalability since all nodes should establish a tree topology among themselves. But FTSP is more scalable since creating a mesh topology network is less complex than tree topology network.

2.5 Data Streaming

In this section few protocols of transport layer for data streaming will be described.

2.5.1 Transport Layer Protocols

Two well-known transport layer protocols exist, transmission control protocol (TCP) and user datagram protocol (UDP) [19]. TCP is connection-oriented which means transmitter will wait for acknowledgment of transmitted packets from receiver. It also supports re-transmission of lost packets and flow control. On the other hand UDP is a connection-less protocol; transmitter does not wait for acknowledgment from receiver so it will send data continuously. This feature made UDP suitable for time-sensitive real time applications.

2.5.2 Message Queuing Telemetry Transport

One of the well-known lightweight protocols for transporting messages between devices is message queuing telemetry transport (MQTT) [20]. An MQTT network consists of a broker and number of clients. Clients publish their messages into the corresponding *topic*. Then broker distributes this information to any other clients which has subscribed to that topic. This simplicity made MQTT a suitable protocol for IoT purposes. Also each connection to the broker can specify a quality of service

in any of following options:

- **At Most Once**: The message is sent only once without waiting for acknowledgement of reception.
- **At Least Once**: The message will be sent periodically until the sender receive acknowledge from receiver.
- **Exactly Once**: The sender and receiver will establish two-level handshake and message will be sent just once.

3
Methodology and Tools

In this chapter the hardware and software tools which we have used will be introduced. Also our methodology and design choices will be explained.

3.1 Hardware

3.1.1 SAM D21 Xplained Pro

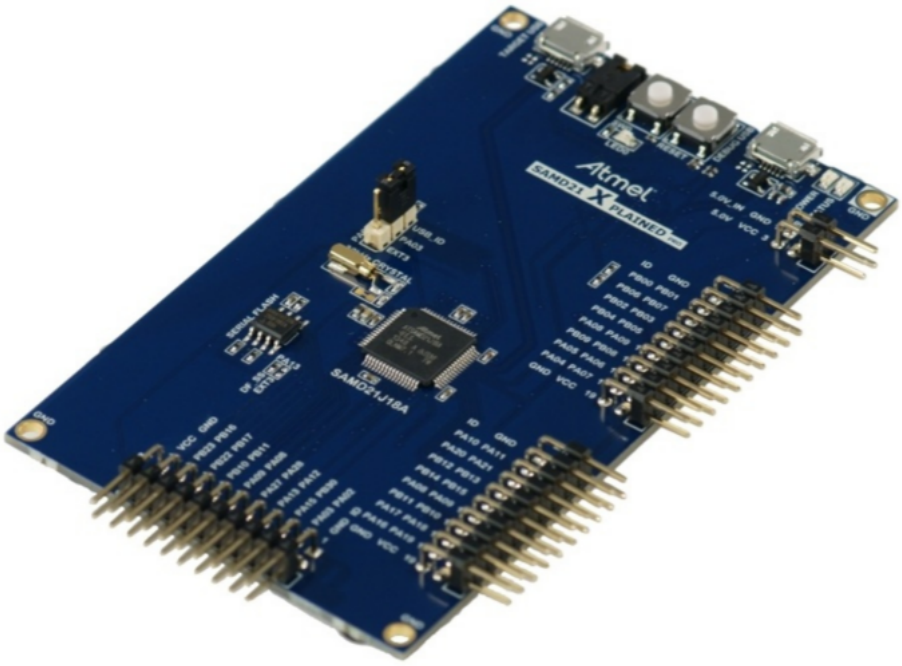

Figure 3.1: SAM D21 Xplained Pro evaluation kit

Our codes were written in C language on "SAM D21 Xplained Pro" (Figure 3.1) evaluation kit from Microchip. This board has ATSAMD21J18A microcontroller [21] which consists of an ARM Cortex-M0+ processor which can operate up to 48 MHz frequency. Flash and SRAM memories are 256 kB and 32 kB respectively. An on-board embedded debugger is also implemented within SAM D21 Xplained Pro which was used to observe the messages from wireless module.

3.1.2 ATWINC3400-XPRO

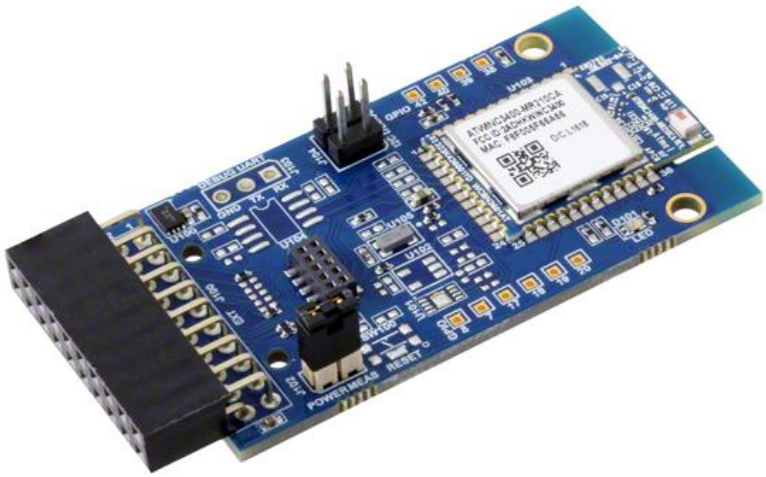

Figure 3.2: ATWINC3400-XPRO

For choosing wireless modules we had a few options from Texas Instruments (TI), Esspresif Systems, and Microchip. The module used in this work is ATWINC3400-XPRO (Figure 3.2) from Microchip since it is quite small and cheap. This board is an evaluation kit for ATWINC3400-MR210CA [22] which can support WiFi and BLE standards. Communication between ATWINC3400-XPRO and SAM D21 is via SPI. Drivers and API are available for this module. These drives let us to set modules as access point or station and implement the communication with socket programming.

3.1.3 NUCLEO-L4P5ZG

NUCLEO-L4P5ZG development board [23] (Figure 3.3) from ST has a microcontroller consisted of an ARM Cortex-M4F processor which can operates up to 120 MHz frequency. Flash and SRAM memories are 1024 kB and 320 kB respectively. This board was used to act as a ReLog and get the time from SAMD21 with SPI interface. An on-board embedded debugger is also implemented within the board which allowed us to run the codes in debug mode to observe the data transferred from SAM D21.

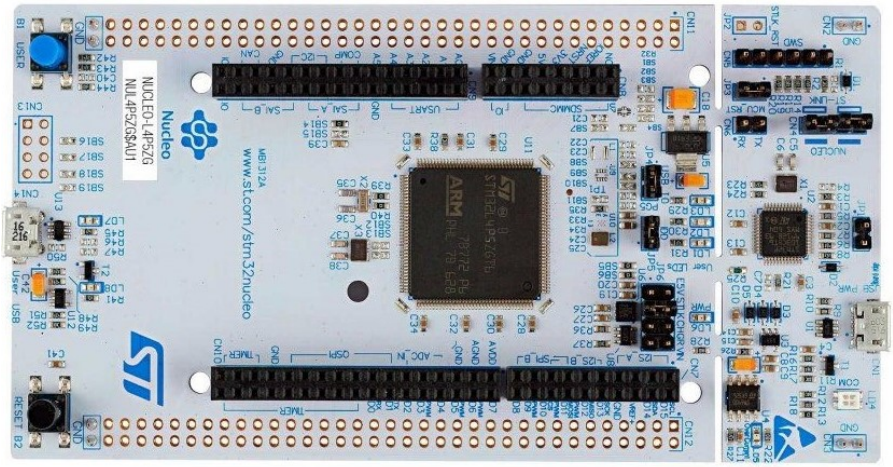

Figure 3.3: NUCLEO-L4P5ZG development board

3.2 Software Tools

In this section the software development tools which were used in our work, will be described.

3.2.1 Microchip Studio

Microchip studio [24] is a software development solution for programming Microchip processors. It also has an advanced software framework (ASF) extension which includes all peripherals and extension modules driver libraries such as ATWINC3400 drivers. It also has a driver for embedded debugger which can connect the host PC to the processor with universal asynchronous receiver/transmitter (UART) interface. The data transmitted over UART can be captured and read by any terminal application such as CoolTerm.

3.2.2 STM32CubeIDE

STM32CubeIDE [25] is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors. This tool allows user to initialize and configure peripherals and middlewares. A handy hardware abstraction layer (HAL) library is also provided which simplifies usage of peripherals.

3.3 Implementation

The embedded core of ReVibe ReLog is similar to STM32L4P5ZGT6U. Unfortunately ST does not produce WiFi modules anymore, hence we had to find a module from another vendor. For this purpose we implemented the time synchronization process within SAM D21 and ATWINC3400 module. Communication between NU-

CLEO and SAM D21 would be handled with SPI interface. The block diagram and prototype of proposed system can be seen in Figures 3.4 and 3.5 respectively:

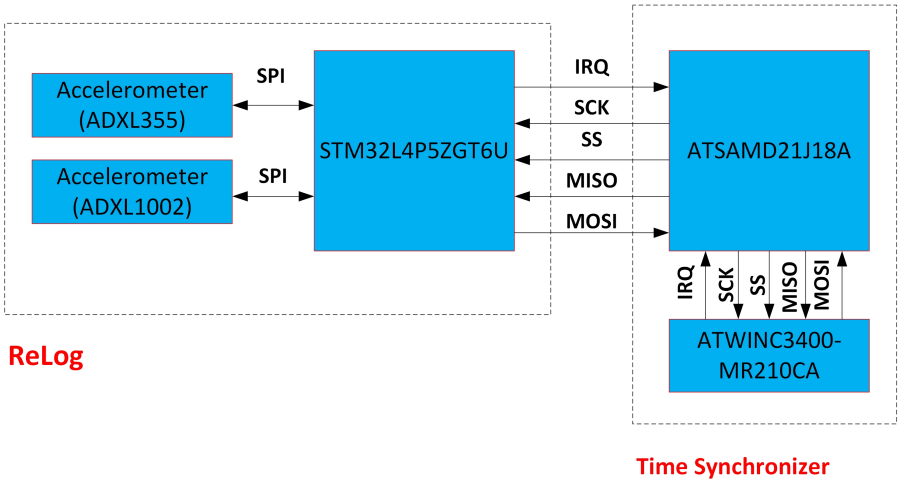

Figure 3.4: System block diagram

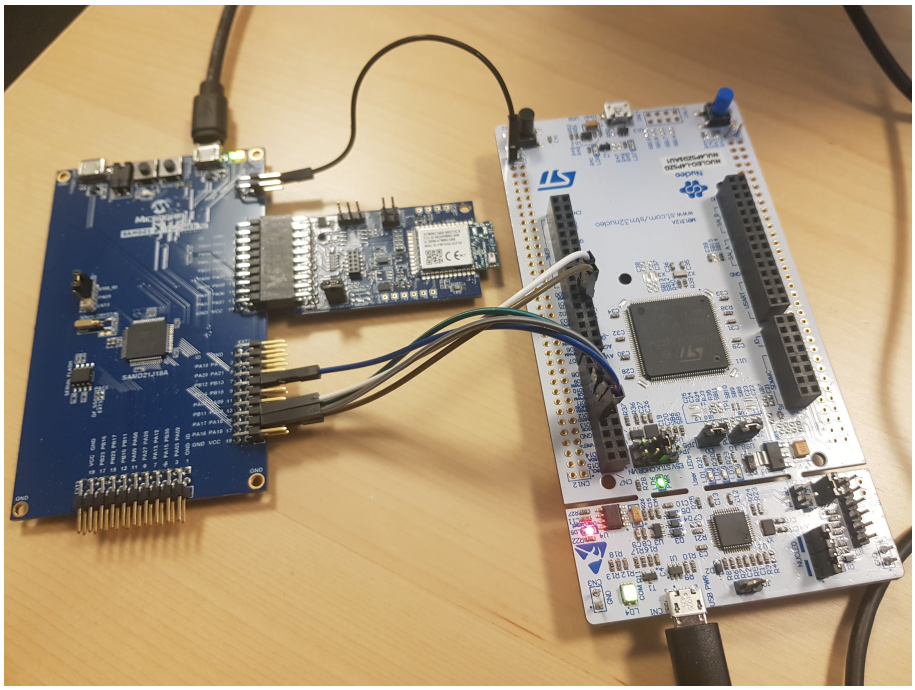

Figure 3.5: Experimental system setup

3.3.1 Time Synchronization System

Timestamping

Since all of the time synchronization protocols require timestamping, a timer should be implemented within processor to keep track of time. SAM D21 comes with different timer peripherals such as real-time clock (RTC), system timer (SysTick), and

timer/counter (TC). Real-time clock is a good option for applications in which a central server needs to know exact dates, hours or minutes. Since in our work we did not require to keep track of time, RTC was not chosen for timestamping. Also RTC does not support sub-second accuracy. System timer works with raising an interrupt. In contrast, timer/counter is a register which is incremented by an user-defined clock. Hence it was a suitable option for timestamping.

SAM D21 supports 8-bit, 16-bit, and 32-bit counters. In our design a 32-Bit counter were used with 32 kHz custom generated clock which means it takes around 37.28 h for the counter to wrap around.

Transport Layer Protocol

Generally UDP is a better option for embedded applications since it has small overhead. Also our application is time-sensitive and real-time. If transmitter tries to send a timestamp packet several times, then timing would be meaningless. In case of missing a packet, the client still synchronize itself with server by next synchronization iteration. Moreover using TCP puts extra pressure on ATWINC3400 buffers. Based on these reasons, we decided to use UDP protocol. The messages which are going to be sent by UDP, will contain the timestamps.

Error and Offset Measurement

For calculating the error and offset of time synchronization protocols, we used *printf* functions for transferring the timestamps to the host PC. These functions send the data over UART interface. We used MATLAB to analyse data and plot the offset errors. To observe the impact of synchronization, both server and client were configured to generate a 1 Hz square wave to see the offset between two waves. In this experiment all *printf* functions were deleted from source codes to reduce delays in code execution as much as possible.

Limitations

Generally time synchronization protocols gradually modify the clock frequency to mitigate the difference between various clocks. But in the SAMD21 board, the clock frequency can be configured before initialization but it cannot be changed afterward. So in our work after calculating the time offset between server and client, this value was added to timer of the client to reduce the offset. In our work we implemented SNTP, PTP, and RBS. We did not test TPSN and FTSP since they required more than two nodes in network. Also in these protocols a node can communicate with several other nodes in the same time which was not possible for us to implement, since WINC3400 can establish a connection with just one other module.

3.3.2 Data Logger

On the data logger side of system, the logger asks for current time from SAM D21 once per each time synchronization interval. During this time the logger continuously samples data from analog-to-digital converter (ADC). In original ReLog the sampler the sampled data comes from two accelerometers but in our tests we did not use any accelerometer, since it was quite tricky to make a connection between them and the processor on breadboard. Instead we used the built-in ADC of STM32 Nucleo. The SPI interface has been used to communicate between STM32 Nucleo and SAM D21.

3.4 Triggers

STM32 has two RTC alarms. One of them was used as start time trigger and the other as the stop time. RTC alarms can be configured in a way which allows them to interrupt the processor minutely, hourly, daily, or at specific date and time. It should be noted that the priority of alarm's interrupt should be lower than real-time operating system (RTOS) interrupts. If this condition is not met, it would lead to the malfunction of RTOS tasks. The start/stop date and time were defined inside the settings file of the ReLog. Also an extra *mode* variable is added to settings which user can choose between periodic logging or logging on specific date and time.

For the implementation of the acceleration triggers, all the stored values inside the buffers were compared to the shock level. When the shock detected, now the values inside the buffer are allowed to be written into the SD card. Also the tasks are designed in way even if no shock is detected during run-time, the remaining values in buffer will be written into the SD card at the termination of logging. We did not change this behaviour since it is a nice way to check if ReLog really worked even when it has not triggered by any shock.

4
Results

In this chapter our experiments and results will be presented. First sections will depict the results of implementation of time synchronization protocols. The rest of the sections will be dedicated to the remote operation features.

4.1 Clock Drift

In the first step we synchronized the client with server just once after initialization to cancel the initialization time difference between them. Then 10,000 time request packets were sent from client. The time interval between each two packets was set to 100 ms. Then response packets from server captured and offset were counted. The results are depicted in Figure 4.1. The slope of offset can be seen clearly in this figure which indicates the clock drift. By extracting the start and end point of slope, the clock drift calculated as 5.475 ppm.

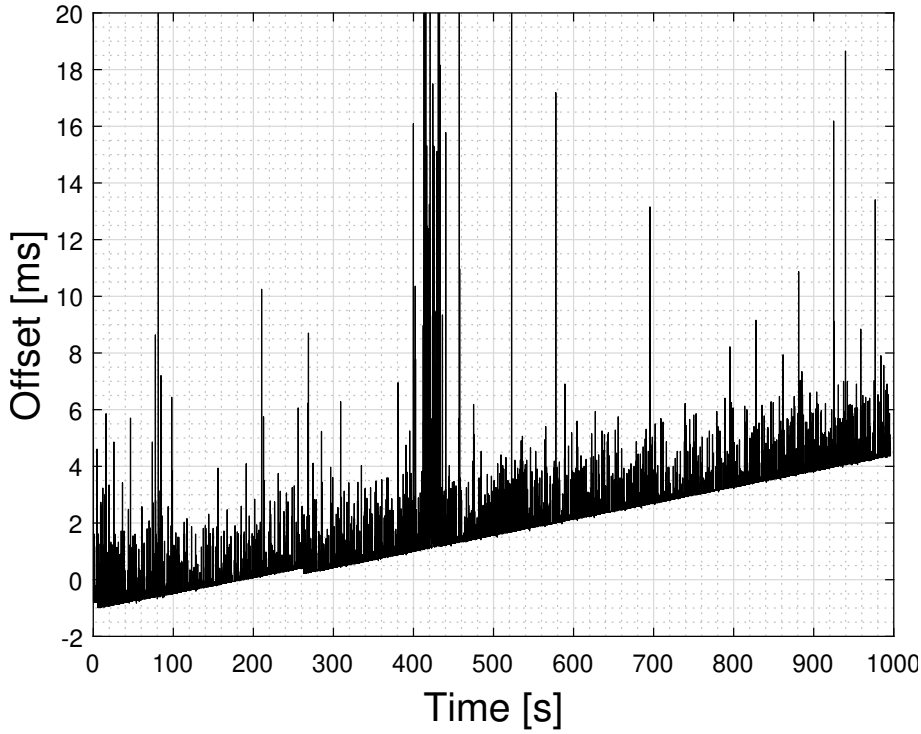

Figure 4.1: Clock drift and offset between server and client

We limited the y-axis to 20 ms to eliminate some of the extreme spikes to see the

slope better. The reason for these spikes in the plot might be because of the random delays of sending packets. Also the reason why the offset is negative in the beginning of experiment is that the client initialized earlier than the server.

4.2 Time Synchronization

After applying the SNTP synchronization, the offset decreased and impact of the clock drift was eliminated as can be seen in Figure 4.2. In this experiment average and maximum offsets were measured 223.025 µs and 7.36 ms respectively. Distribution of offset error is also depicted in Figure 4.3. Based on this results standard deviation of offset was calculated as 850 µs. We should not that for calculating the standard deviation, first ten set of calculated offsets has been ignored since they are relatively large due to initialization time difference.

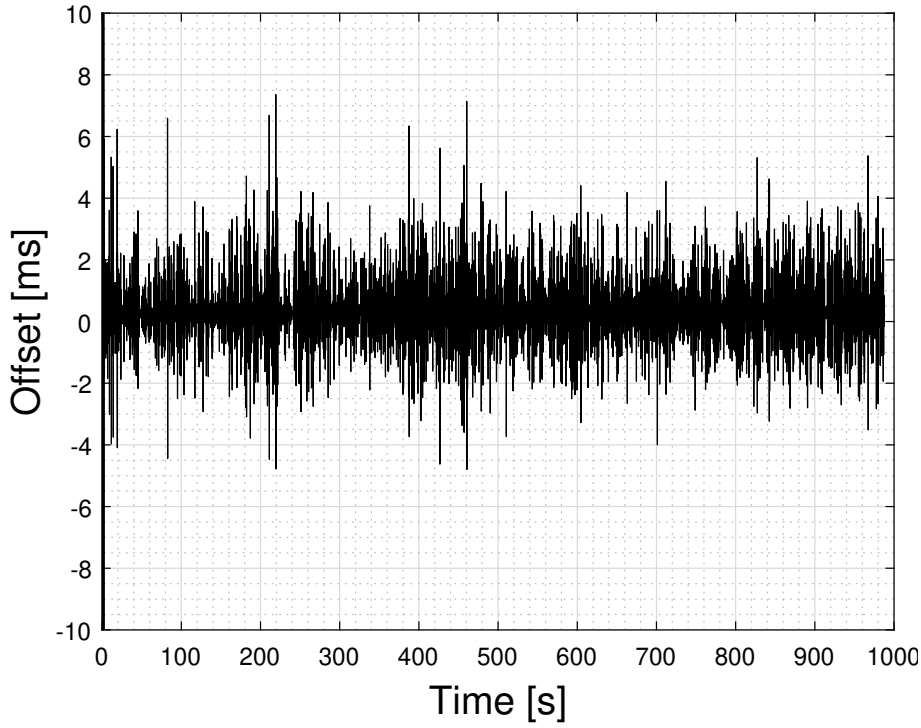

Figure 4.2: Offset between server and client after applying SNTP

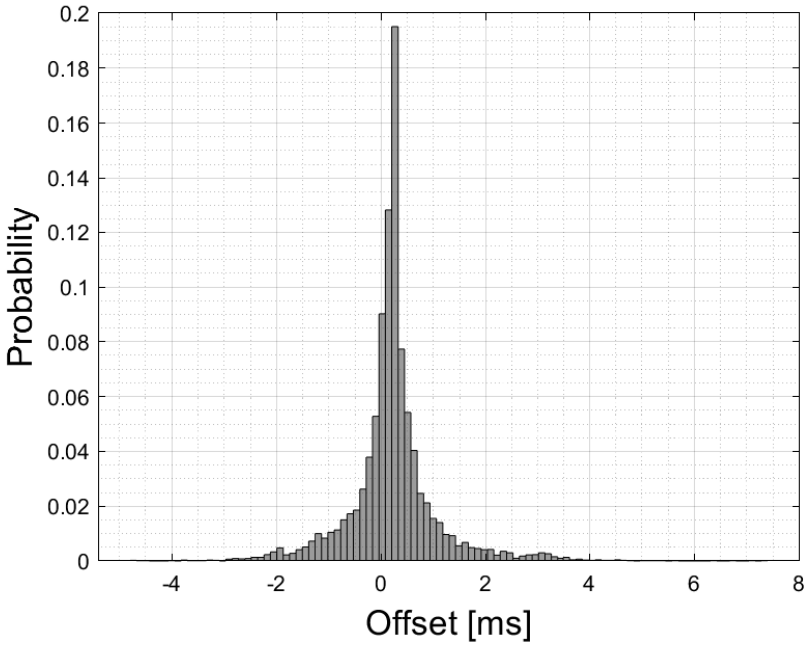

Figure 4.3: Distribution of offset between server and client after applying SNTP

In an experiment with PTP, the client send a dummy packet to the server to ask for synchronization. Then after receiving sync response from server, the client sends the delay request to the server. The time interval between sending dummy packet and delay request is set to 100 ms. So each PTP synchronization takes 200 ms in total. The results can be seen in Figure 4.4. In this experiment average and maximum offset were measured 223.625 µs and 57.075 ms respectively. Distribution of offset error is also depicted in Figure 4.5 and standard deviation of offset was calculated as 5.1 ms. We should note that this figure just focused on the offset values around 0. Data corresponding to the spikes in Figure 4.4 were not shown in histogram since the probability of spike occurrence is too low (around 0.006). If we also remove spikes from offset, then we get standard deviation of 975 µs.

For implementing RBS, we set the server to send four reference packets with 100 ms time interval between each reference. In total server sent 10,000 packets to the client which led to 2,500 sets of data for synchronization. The results can be seen in Figure 4.6. In this experiment average and maximum offsets were measured 224.9 µs and 10.52 ms respectively. Distribution of offset error is depicted in Figure 4.7 and standard deviation of offset was calculated as 922 µs.

Although the synchronization intervals of each of implemented protocols are different (100 ms for SNTP, 200 ms for PTP, and 400 ms for RBS), we did not try to set an equal interval for all of them. This is because we wanted to find the best and shortest possible interval for each of these protocols to achieve highest timing accuracy. Also in all experiments, still some random long delays exist during transmission. Hence several spikes in Figures 4.2, 4.4, 4.6 can be seen.

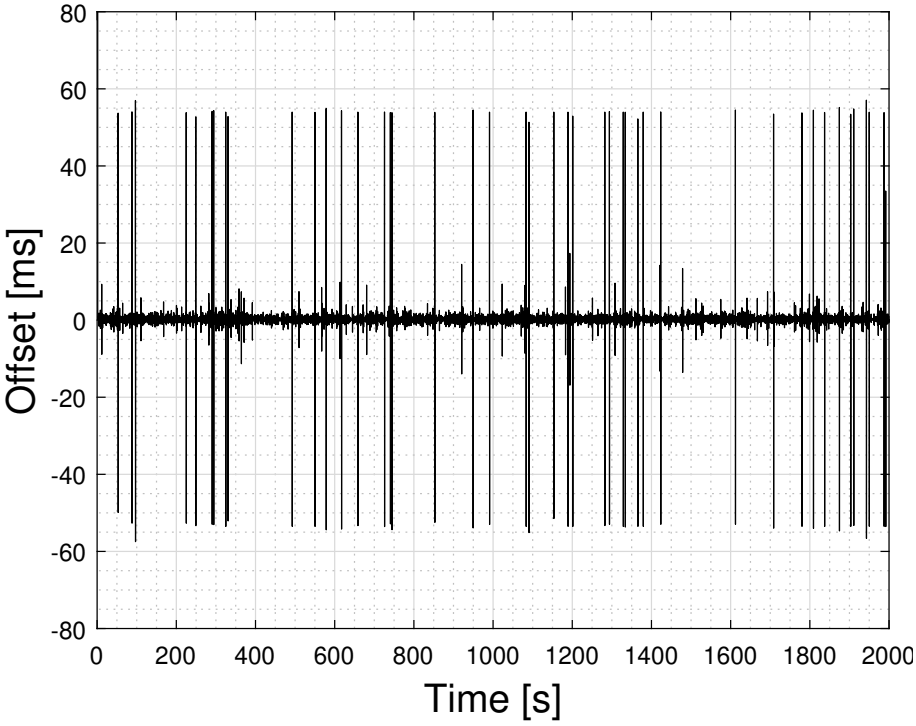

Figure 4.4: Offset between server and client after applying PTP

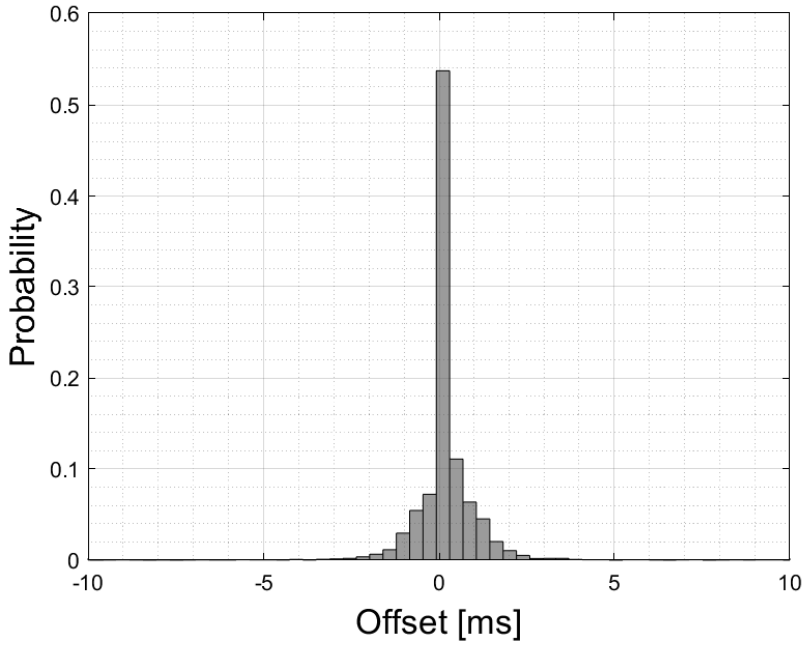

Figure 4.5: Distribution of offset between server and client after applying PTP

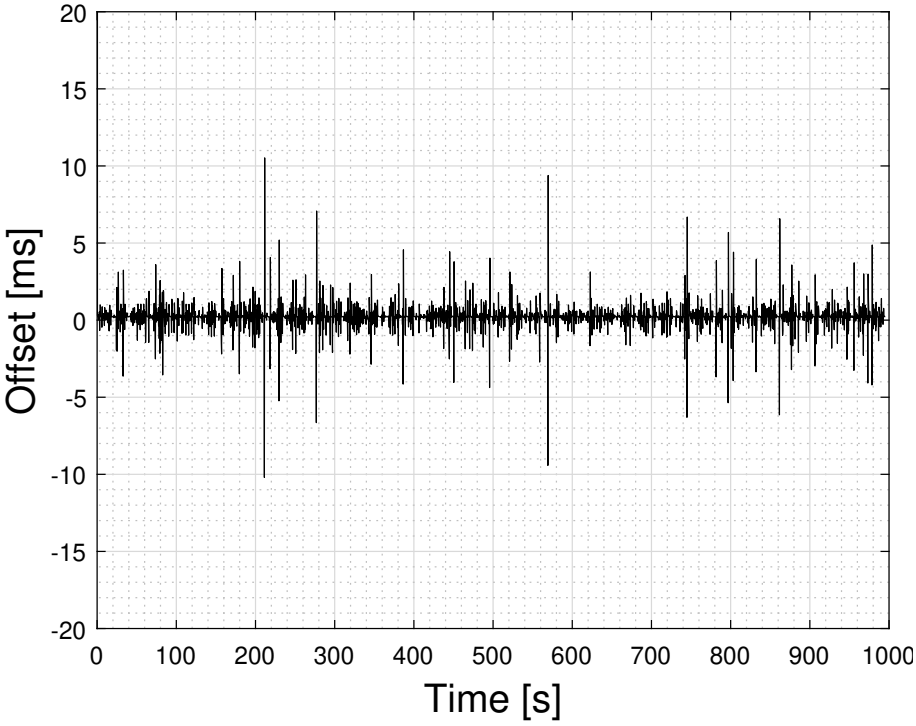

Figure 4.6: Offset between server and client after applying RBS

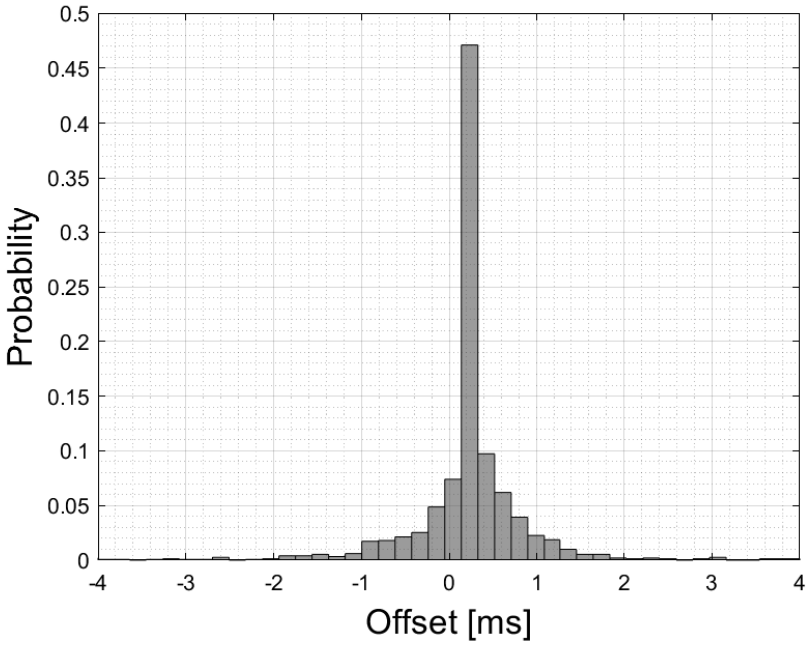

Figure 4.7: Distribution of offset between server and client after applying RBS

4.3 Comparison of Protocols

The comparison between these implemented protocols can be seen in Table 4.1. SNTP and PTP have same average error around 223 µs but PTP suffers from maximum error of 57 ms which is 8 times larger than maximum error of SNTP. When it comes to RBS, it has a bit larger average error and maximum error is also higher than SNTP. Moreover in terms of error standard deviation, SNTP has lowest value. So all these reasons led us to choose SNTP as most time-accurate protocol and use it for further steps.

Table 4.1: Comparison between time synchronization protocols

Protocol	Average Error (µs)	Maximum Error (ms)	Standard Deviation (ms)
SNTP	223.025	7.36	0.85
PTP	223.625	57.075	5.1
RBS	224.9	10.52	0.92

We also have tested SNTP for 1 Hz square wave generation to observe the synchronization on waveform generation. The waveforms of server (Pink) and client (Blue) can be seen in Figure 4.8. The offset between the two square waves was mostly below 1 ms in each period of signals but occasionally it increased up till 3 ms. This behaviour was expected since there were some spikes in offset calculations as can be seen in Figure 4.2.

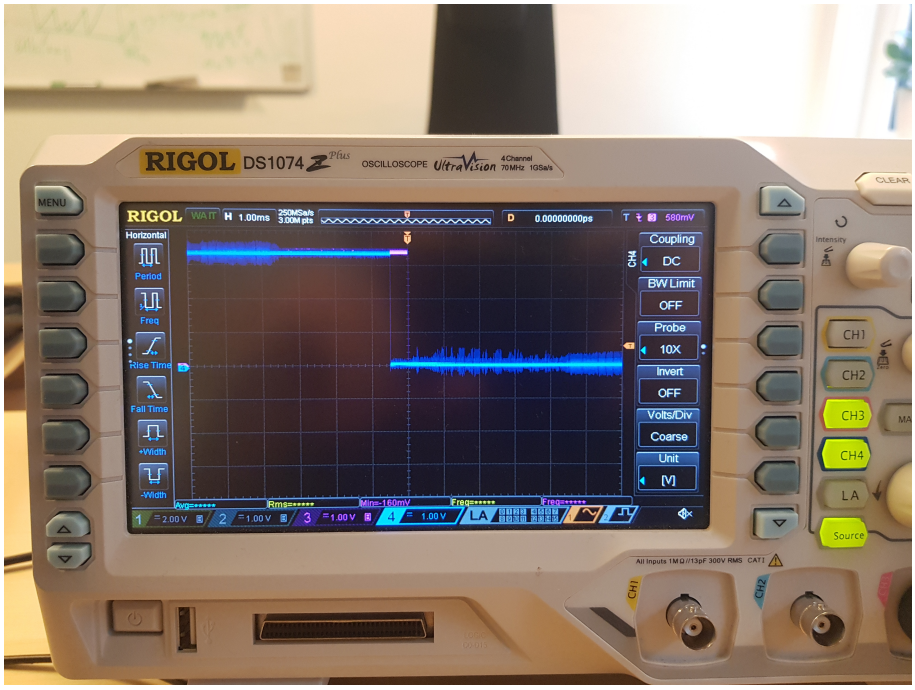

Figure 4.8: Synchronized square waves with SNTP

4.4 Time Synchronization for Data Logger

In this experiment we generated a 1 Hz sine wave from the oscilloscope with 1.3 V amplitude and 1.2 V offset. This signal was fed into 8-bit ADCs of two STM32 Nucleo board each of which was connected to a SAM D21. STM32 asks for the timestamp from SAM D21 and then captures ten samples from ADC with 10 ms interval. The captured data and also timestamps were transferred from STM32 to host PC with UART interface. Since for each ten samples one timestamp has been stored, the timestamp was used as capturing time of first data and for remaining 9 samples 10 ms were added to previous captured time. The sampled wave from client and server can be seen in Figure 4.9.

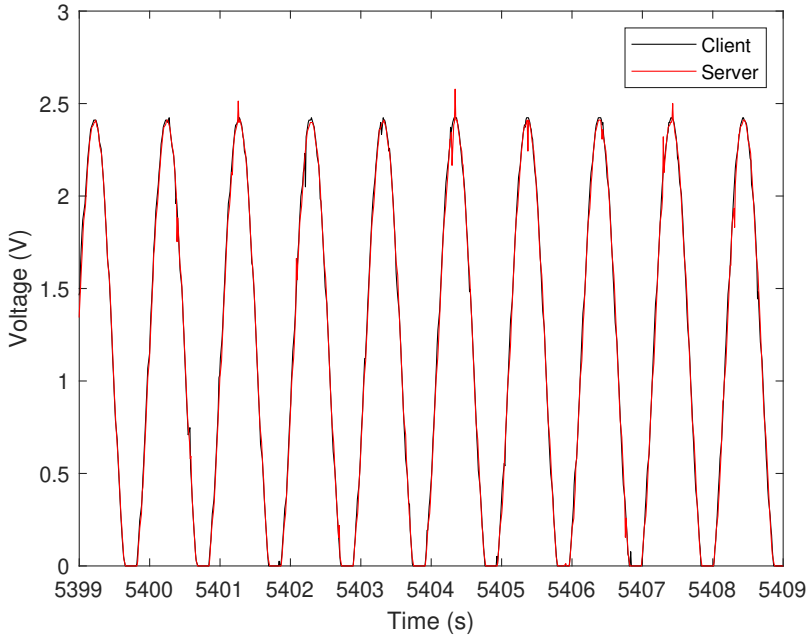

Figure 4.9: Captured sine waves from two synchronized STM32 Nucleo boards

As can be seen in Figure 4.9, the x-axis of the plot is time which has been obtained by converting timestamps to real time values. The waveforms are synchronized and perfectly matched. A pre-defined function from MATLAB central file exchange[28] has been used to fit a sine curve into captured data. For each sampled wave we had:

$$\sin_{\text{client}} = 1.1476 + 1.2558 \sin(2\pi \times 1.114 + 3.7987) \tag{4.1}$$

$$\sin_{\text{server}} = 1.1376 + 1.2528 \sin(2\pi \times 1.1106 + 3.7124) \tag{4.2}$$

As can be seen in 4.1 and 4.2, the sine curves are almost accurate and the phase shift between two waves is 0.0863 rad which is equal to 14 ms. Also we used the cross correlation between two signals to obtain the similarity between two waveforms. The peak in the cross-correlated occurred in point 1 as it is depicted in Figure 4.10.

Since the time interval between each data point in both signals is 10 ms, then the offset between signals is 10 ms.

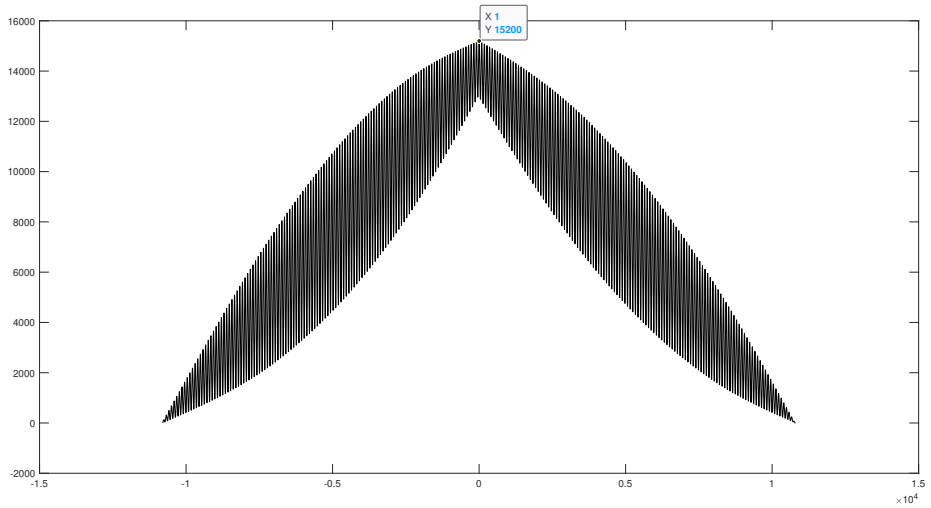

Figure 4.10: Cross correlation between captured signals

4.5 Data Streaming

In the first step, we tried to connect one wireless module to a custom access point in the office to be able to connect to a cloud service. We used *PubNub* [26] cloud in this experiment. But the problem with sending data to this cloud service was, PubNub just supports *JavaScript object notation (JSON)* [27] format. JSON stores and transmit data objects in format of attribute–value pairs. Hence it reduces the amount of data which can be sent in a single packet.

In the next experiment we used another ATWINC3400 module as a receiver. We were able to send 1200 bytes of data each 5 ms. That means data transfer rate is 240 kB s^{-1} which is enough to download a 1 GB from the SD card in 70 minutes. Although the TCP protocol is safer solution for data streaming, but it puts extra pressure on the transmitter-receiver buffer of wireless module. Hence we used UDP for data streaming although chance of loss of some data packets exists.

4.6 Triggers

The time triggers worked perfectly and ReLog can start/stop logging at specific date and time or periodically. When it comes to acceleration triggers, ReLog could detect the shocks and start writing data into the SD card. Since after detection of shock the whole values of buffer will be transferred to the SD card, data during few moments before shock would also be written into SD card.

We put ReLog in running state for one minute. In the first experiment no shock was applied on ReLog. As can be seen in Figure 4.11 only around one second of run-time has been written into the SD card which is the remaining data inside buffer after termination of running. In the second experiment again we logged data for one minute but we shook ReLog strongly in the last seconds. The results can be seen in Figure 4.12 around 5 seconds before shock detection has been captured due to buffers behaviour.

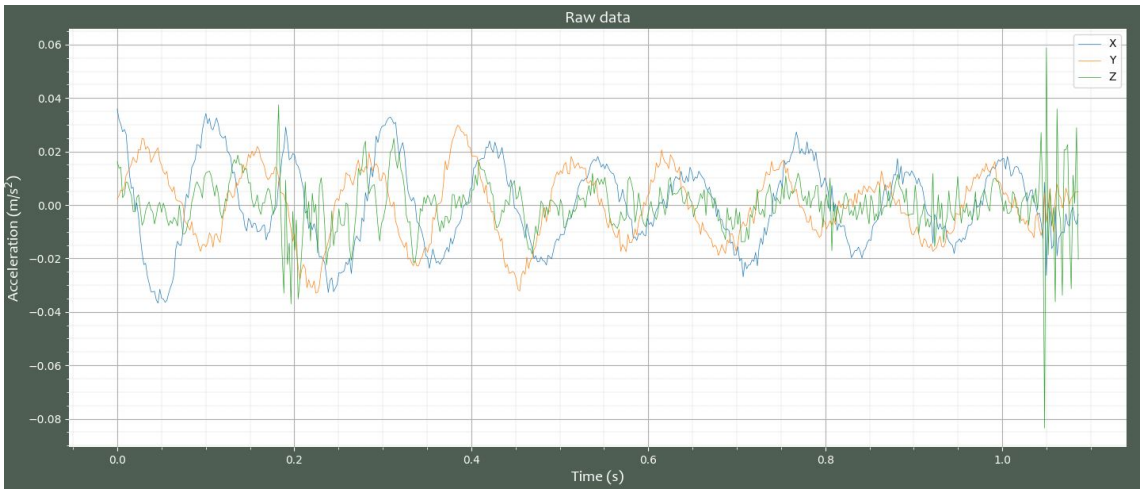

Figure 4.11: Vibration log without applied shock

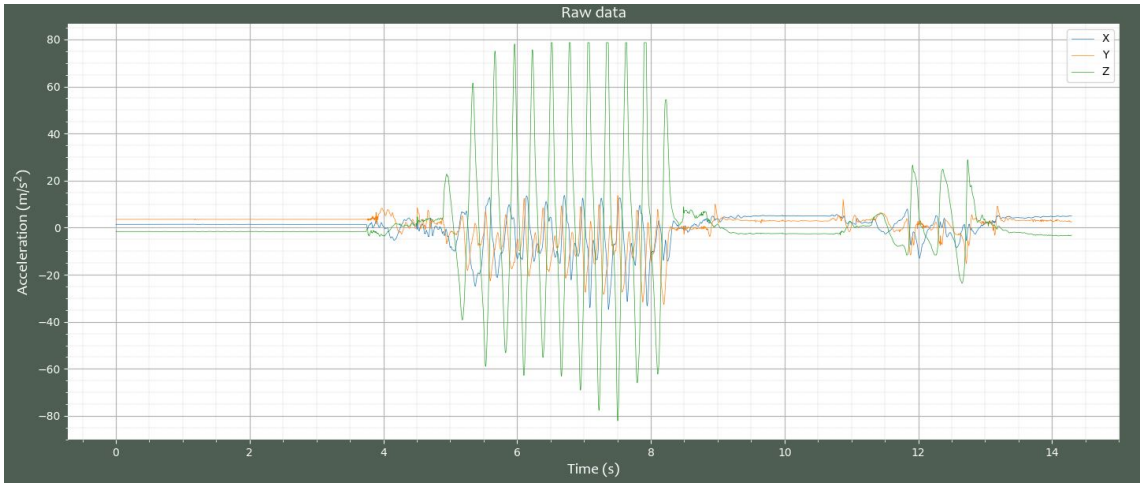

Figure 4.12: Vibration log with applied shock

5
Discussion and Conclusion

5.1 Discussion

5.1.1 Transport Layer Protocol

As we mentioned before, we used UDP which is a connection-less protocol. That means it does not wait for acknowledgment from the receiver. In our various experiments, in which we sent 10,000 packets, we received between 9,960 to 9,990 packets. That means 0.1 - 0.4 % of packets were lost during transmission. Even if some packets get lots, devices could still synchronize themselves in upcoming periods.

5.1.2 PTP Experiment Results

As can be seen in Figure 4.4 there were some spikes in offset error in contrast with results from SNTP experiment. The reason might be the difference between transmitting and receiving speed. In the SNTP experiment, the client sends one timestamp and receives two timestamps within one transmission from the server. But in the PTP experiment, the client sends two messages (one dummy message and one timestamp) and receives two timestamps in two separate transmission from the server. So PTP requires more data transfers between the client and the server. In general, receiving packets puts more pressure on transmission buffers and takes more time than sending. Hence, this suffers more from delays. These delays would affect protocols which require more data transfers.

As we mentioned before, PTP is one of the most accurate synchronization protocols since it relies on hardware-based timestamping. But since ATWINC3400 drivers do not support any access to data link and physical layers, we could not take advantage of hardware timestamps. That is the reason we did not achieve the highest accuracy with PTP.

5.1.3 Offset

In the square wave generation and logging sine wave experiments, the client was always 100 ms behind the server. The reason most likely is because of timer/counter behaviour in SAM D21. As we mentioned in previous chapters, we set the synchronization interval to 100 ms since in shorter intervals the buffer of WiFi module was getting full which led to data loss. For implementing the waiting time we used *delay* function. This function puts CPU and timers into sleep mode. So for 100 ms

timer value would be frozen which causes a constant offset between the client and the server.

As we saw in section 4.3, we achieved accuracy below 1 ms with SNTP. But when we added the logger ADC, the accuracy we got was 10 ms. The reason might be due to behaviour of SPI interface between logger and time synchronizer. We set timeout duration of 10 ms for the SPI receive function. Shorter intervals led to corruption of data after a while. But with 10 ms timeout no data corruption happened. So it seems that the SPI interface between logger and time synchronizer is not fast enough. So it will kill the accuracy which we got in section 4.3.

5.1.4 Data Streaming Rate

The data transfer rate we achieved was 240 kB/s. This rate was obtained by sending 1200 bytes in 5 ms intervals. But as we experienced in time synchronization experiment, any syncing interval below 100 ms causes the buffer to get full. The reason that we could send data in shorter intervals in data streaming experiment, was we were just sending packets to the server and not receiving any packet from it. But in synchronization experiments we expected data from the server which increased the pressure on the buffer.

5.1.5 Integration into the Main Product

All works we did in this thesis was trying to build a prototype for integrating time synchronization into ReLog. But this feature cannot be implemented into the main product currently. One of the main problems is that the current case of ReLog is made of aluminium which shields the wireless signals. So the first step for integrating wireless module into ReLog is to design a new case with an alternative material.

Another issue with the proposed system is that the ReLog's processor and wireless module are not produced by a same vendor. Hence we had to use another processor from Microchip to be able to establish communication between them. So this prototype has an extra processor which does nothing except timestamping and send timestamps to the wireless modules and ReLog. That would cost as area overhead in final PCB of product.

The best solution to avoid using an extra processor would be to find another wireless module from ST. Unfortunately ST does not produce WiFi solutions anymore, but it provides several BLE modules. For time synchronization purpose BLE would be efficient since data transmission between units is just limited to the timestamps. But for other purposes such as data streaming, BLE would not be an efficient solution. Another benefit of using a module from ST is that the timer can be implemented within the STM32 processor and there is no need to transfer timestamps over SPI which decreases the timing accuracy.

For data streaming, although we achieved high data transfer rate, the communica-

tion is still between two wireless modules. That would limit the use case in a small area. The suitable solution is to connect the wireless module to a random access point in area and then send the data to a cloud server. One of the possible cloud servers for this purpose could be *Amazon web service (AWS)* since it can support sending up to 128 kB in one MQTT message. AWS has provided APIs for some hardware platforms which are widely used such as ESP32 but ATWINC3400 was not in list of supported kits. Hence for future implementation of data streaming into ReLog another module (e.g. ESP32) might be used which has support from AWS.

Another scenario for the future is that the data streaming function might be implemented into an stand-alone module which can communicate with ReLog by universal serial bus (USB) interface. In this case there is no need for re-designing the case and PCB of ReLog and stand-alone module can be directly connected to the ReLog.

5.1.6 Triggers

The time triggers were tested and they are going to be implemented into the firmware of the main product which is for sale. But acceleration trigger is still not complete enough to be part of the commercial firmware. Sometimes some wave files cannot be read by VinInspect since it cannot reshape the data array into three columns for each axis. In these cases one or two redundant data sample can be added to the end of wave file. Then these files will become readable for VibInspect. But it is better to fix this issue inside the microprocessor's code rather than in VibInspect.

5.2 Conclusion

Remote operation and connectivity is one of the most beneficial features of IoT systems. Some of these useful futures are time synchronization, data streaming and triggers. In this work we tried to add these functionalities to the ReLog data logger.

As we mentioned in previous chapters, several time synchronization protocols exist for WSNs. In this work we just focused on synchronization of two units and implemented SNTP, PTP and RBS protocols. The highest timing accuracy was achieved with SNTP and it was 10 ms which is enough for sampling rates up to 50 Hz. We also investigated the possibility of data streaming over WiFi connection. But all these features were implemented on the prototypes. Finally we implemented time and acceleration triggers into the ReLog's firmware.

Bibliography

[1] F. Tirado-Andrés, A. Rozas, A. Araujo, "A Methodology for Choosing Time Synchronization Strategies for Wireless IoT Networks", *Sensors*, vol. 19, 2019.

[2] X. Liu et al., "Overview of Spintronic Sensors With Internet of Things for Smart Living," *IEEE Transactions on Magnetics*, vol. 55, no. 11, pp. 1-22, Nov. 2019.

[3] ReLog vibration data logger. Rev. 002. ReVibe Energy. 2020

[4] *VibInspect*. (2020). ReVibe Energy. Available: https://revibeenergy.com/vibinspect-analysing-software/

[5] R. Prakash, K. Nygard, "Time Synchronization in Wireless Sensor Networks: A Survey", *International Journal of UbiComp*, 2010.

[6] K. Hilmersson, F. Gummesson, "Time Synchronization in Short Range Wireless Networks", Department of Electrical and Information Technology, Lund University, 2016.

[7] NINA-B1 series. UBX-15019243. Rev. 14. u-blox. October 2019

[8] T. Sheng, K. Jing, L. Zunzun, D. Pengfei, "Design and Implementation of Time Synchronization Experimental System for Wireless Sensor Networks", *International Journal of Online Engineering (iJOE)*, vol. 14, p. 212, 2018.

[9] nRF24L01 Single Chip 2.4GHz Transceiver. Rev. 2.0. Nordic Semiconductor. July 2007

[10] M. Hicham, M. Dagenais, "Internal Clock Drift Estimation in Computer Clusters", *Journal of Computer Systems, Networks, and Communications*, 2008.

[11] Y. Li, D. Li, W. Cui and R. Zhang, "Research based on OSI model," *IEEE 3rd International Conference on Communication Software and Networks*, Xi'an, China, pp. 554-557, 2011.

[12] V.K. Garg, "IEEE 802.11 WLAN" in *Wireless Communications and Networking*, Elsevier, 2011.

[13] O. Seijo, J. López-Fernández, H. Bernhard, I. Val, "Enhanced Timestamping Method for Sub-Nanosecond Time Synchronization in IEEE 802.11 over WLAN Standard Conditions", *IEEE Transactions on Industrial Informatics*, 2020.

[14] P. Rybaczyk, *Expert Network Time Protocol: An Experience in Time with NTP*, 1st ed. Apress, 2005

[15] A. Garg, A. Yadav, A. Sikora and A. S. Sairam, "Wireless Precision Time Protocol," in *IEEE Communications Letters*, vol. 22, no. 4, pp. 812-815, April 2018

[16] K. Tarnay, G. Adamis, T. Dulai, "Reference Broadcast Synchronization Protocol" in *Advanced Communication Protocol Technologies - Solutions, Methods, and Applications*, IGI Global, 2013

[17] J. Cui, Y. Liu, K. Wei, H. Cao, "Research on Clock Source Correction Method based on Wireless Sensor Network and TPSN Network Protocol", *Journal of Physics: Conference Series*, 2020.

[18] Y. Hlaing and N. A. Maung Maung, "Hybrid Time Synchronization for ZigBee Networks: An Empirical Approach," *2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, Phuket, Thailand, pp. 376-379, 2020.

[19] M. T. Naing, T. T. Khaing and A. H. Maw, "Evaluation of TCP and UDP Traffic over Software-Defined Networking," *2019 International Conference on Advanced Information Technologies (ICAIT)*, 2019, pp. 7-12

[20] S. Quincozes, T. Emilio and J. Kazienko, "MQTT Protocol: Fundamentals, Tools and Future Directions," in *IEEE Latin America Transactions*, vol. 17, no. 09, pp. 1439-1448, September 2019

[21] SAM D21/DA1 Family. DS40001882F. Rev. F. Microchip Technology Inc. March 2020

[22] ATWINC3400-MR210xA. Microchip Technology Inc. December 2020

[23] STM32 Nucleo-144 boards (MB1312). UM2179. Rev. 9. STMicroelectronics. November 2019

[24] Microchip Studio User Guide. DS50002718C. Rev. D. Microchip Technology Inc. December 2020

[25] Integrated development environment for STM32 products. DB3871. Rev. 4. STMicroelectronics. November 2020

[26] "PubNub: Real-Time In-App Chat and Communication Platform". https://www.pubnub.com/

[27] A. A. Abd El-Aziz and A. Kannan, "JSON encryption," *2014 International Conference on Computer Communication and Informatics*, 2014, pp. 1-6

[28] P. Seibold (2021). Sine fitting, MATLAB Central File Exchange. Retrieved March 19, 2021.