BACHELOR'S THESIS 2022

Wayfinder - Development of a path indication prototype module using LEDS and graphs

Erik Andersson & Robert Bäckström

ISSN 1651-2197
LU-CS/HBG-EX: 2022-12

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

Wayfinder

- Development of a path indication prototype module using LEDS and graphs

LUND UNIVERSITY
Campus Helsingborg

LTH School of Engineering at Campus Helsingborg
Department of Computer Science

Bachelor thesis:

Erik Andersson

Robert Bäckström

© Copyright Erik Andersson, Robert Bäckström

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Lunds universitet
Lund 2022

Abstract

This bachelor thesis was conducted in cooperation with Axis Communications AB and the authors are Computer Engineering students at Lund University, Campus Helsingborg.

All clocks drift due to that the frequency of which the crystal that is responsible for the system clock's timestamp changes over time. To counter this there are two well-known protocols, NTP and PTP, that can be utilized to resynchronize the clocks. This study aims to examine which protocol in the aspect of cost, security and precision is suitable for LED synchronization on the Axis network speaker C1410. Furthermore, the study seeks to find a technical solution for how to indicate a path with the LEDs on multiple devices and thus expanding the use cases for the built-in LEDs.

The result of the thesis is a prototype that utilizes NTP for time synchronization. Furthermore, the solution to indicate a path builds upon applied graph theory, where offset for the LEDs on the devices depends on their relative position to each other. Additionally, an IoT message protocol, more specifically MQTT was used to handle distribution of data between the devices. Finally, an API and a GUI was developed to send LED and graph data from an external device, that is, a device that is not in the IoT messaging scheme.

Keywords: NTP, PTP, offset, synchronization, LED, IoT

Sammanfattning

Den här kandidatuppsatsen gjordes i samarbete med Axis Communications AB och författarna är dataingenjörsstudenter på Lunds universitet, Campus Helsingborg.

Alla klockor har klockdrift på grund av att frekvensen för kristallen som är ansvarig för systemklockans tidsstämpling ändras över tid. För att motverka detta finns det två välkända protokoll, NTP och PTP, vilka kan användas för att synkronisera klockorna. Denna studie syftar till att undersöka vilket av dessa protokoll i aspekterna av kostnad, säkerhet, strömförbrukning och noggrannhet som lämpar sig för synkronisering av lysdioder på Axis nätverkshögtalare C1410. Vidare försöker studien att hitta en teknisk lösning för hur man indikerar en väg med hjälp av lysdioderna på flera av högtalarna och således utöka användningsområdet för de inbyggda lysdioderna.

Det slutgiltiga resultatet av kandidatuppsatsen är en prototyp som använder NTP för tidssynkronisering. Vidare så bygger lösningen för hur man indikerar en väg med lysdioderna på tillämpad grafteori, där offset för lysdioderna beror på högtalarnas relativa position till varandra. Dessutom användes ett IoT meddelandeprotokoll, mer specifikt MQTT för att hantera distribution av data mellan enheterna. Slutligen utvecklades ett API och ett grafiskt gränssnitt för att skicka data rörande lysdioderna och grafen från en extern enhet, det vill säga, en enhet som inte ingår i IoT-meddelande systemet.

Nyckelord: NTP, PTP, offset, synkronisering, lysdioder, IoT

Acknowledgements

Firstly, we would like to express our gratitude to our supervisor at Lund University, Christian Nyberg, for the received feedback and support we have gotten along the process. As well as our examinator Marcus Klang for giving the final feedback to our thesis. We would also want to extend our gratitude to, and thank our supervisor at Axis Communications AB, Arvid Nihlgård Lindell, for always helping and pointing us in the right direction. Our gratitude and thanks also goes out to our manager, Duja El-Khamisi, for making it possible for us to conduct our thesis at Axis Communication AB. Finally, we would like to thank all the people at New Business that took the time to help us as well as our family and friends for their support.

List of contents

1 Introduction ... 1
1.1 Background ... 1
1.2 Goals and purposes ... 2
1.3 Defining the problem ... 2
1.4 Limitations ... 3
2 Technical background ... 5
2.1 Time ... 5
2.1.1 Clock Offset ... 5
2.1.2 Clock Skew ... 5
2.1.3 Clock Drift ... 6
2.2 Time Synchronization Protocols ... 6
2.2.1 NTP ... 6
2.2.2 PTP ... 9
2.3 MQTT ... 12
2.3.1 The Broker ... 12
2.3.2 Broker-Client Connection ... 12
2.3.3 Publish And Subscribe ... 13
2.3.4 Topics ... 15
2.4 Build system ... 15
2.4.1 Bitbake ... 15
2.4.2 Meson ... 16
2.5 JSON ... 16
2.5.1 JSON Object ... 16
2.5.2 JSON Array ... 17
2.6 API ... 17
2.6.1 Apache HTTP Server ... 17
2.6.2 SOAP ... 18
3 Method ... 19
3.1 NTP Experiment ... 20
3.2 Device to Device Communication ... 21
3.3 Patterns ... 22
3.4 API ... 22
3.5 GUI ... 23
3.6 Source Criticism ... 23
4 Results ... 27
4.1 NTP Server Synchronization ... 27
4.2 Time Offset Device – Device ... 29
5 Analysis ... 33
5.1 Selecting Time Synchronization Protocol ... 33

5.1.1 Precision 33
5.1.2 Cost 35
5.1.3 Verdict..35
5.2 Selecting IoT Messaging Protocol 36
5.3 Data For Synchronization 36
5.3.1 Graph data 37
5.3.2 LED data...37
5.4 External Communication with a Device 38
6 Implementation 39
6.1 Device-Device Communication 39
6.2 Graph 42
6.3 Patterns 43
6.3.1 Flash 43
6.3.2 Pulse 45
6.3.3 Sine...47
6.4 API...49
6.5 GUI 52
6.6 Overview How it works 52
7 Conclusion 54
7.1 Ethical Aspects..56
8 Future Development & Research 56
9 Terminology 58
10 References 60
11 Appendix 64

1 Introduction

1.1 Background

Axis Communications AB is a Swedish company founded in 1984 with headquarters in Lund. Furthermore, Axis employs more than 3800 people in fifty countries [1]. The company is the market leader in network video [2] and offers further solutions in other areas such as access control, intercom, and network audio [1].

Axis Communications has expressed an interest to investigate how the AXIS C1410 Network Mini Speaker with its built-in LEDs and passive infrared sensor (PIR) could be synchronized over an IP network to be used in new ways. Suggested areas of use could be to lead individuals in a specific direction or to reconstruct a path taken by an individual that has been detected by the speaker's sensors [4]. A concrete example could for example be to, during a crisis, be able to use the speakers to communicate an escape route with both the speaker's built-in LEDs and microphone. This implies furthermore, that several speakers are required to be synchronized to allow for an arbitrary pattern to be constructed.

Axis network speaker devices run on a Linux-based system however limited in memory to 256mb and processing power compared to a desktop Linux machine. This means that relatively conventional user-space[1] applications can be developed to the speakers in C, C++, Rust, or other similar compiled languages. Exceptions, however, apply to languages that require greater runtime such as Java or C#. Some of these speakers, like the C1410 model, have built-in components such as LEDs that can be controlled by software running on the devices as well as sensors that can detect movements (PIR). Furthermore, the speakers have support for two-way communication with the built-in microphone and can be connected to an IP network as well as powered with POE[2] technology.

According to Tirado-Andrés, Rozas, and Araujo [3], there is no general synchronization strategy that can be applied to an arbitrary application. The authors [3], on the other hand,

---

[1] All the code that runs outside the kernel of the operating system.

[2] POE is an acronym for power over ethernet and it allows network cables to also supply power.

report that the requirements that exist on the application can be broken down into parameters[3] where the weight of each parameter is weighed in to find a suitable strategy. Furthermore, according to [3], these parameters depend more or less on each other and consequently, it is not possible to maximize each parameter. Different synchronization strategies can thus be examined concerning the parameters presented by Tirado-Andrés, Rozas, and Araujo [3] as well as the requirements placed on the application. Furthermore, appropriate strategies can be further explored as to whether these can be modified to better meet the parameter values.

Suggestions for synchronization protocols to start exploring may be Network Time Protocol (NTP) and Precision Time Protocol (PTP). Further, we thought we would look at different methods such as synchronous ethernet and Berkeley's algorithm. These can be implemented in the system to be further evaluated in terms of cost, security and precision.

1.2 Goals and purposes

This thesis will examine how to best synchronize light and audio between different devices in a network to be able to produce arbitrary synchronized patterns with LEDs in conjunction with audio, from multiple devices. A prototype of the system that performs the synchronization shall be developed for the Axis speaker system. The prototype will then be evaluated according to the criteria that were described in the previous subsection. The expected result is to be able to expand the area of use for the built-in components (PIR sensors and LED lamps) as well as find out the most suitable method of synchronization, given the system we are working with.

1.3 Defining the problem

The problem this thesis should solve is to 1) find the most suitable method of synchronization, given the limitations and criteria we have. 2) Present the findings in a comprehensible way. More precisely, the study seeks to address the following questions:

---

[3] cost, energy consumption, clock stability, security, hardware requirements, software abstraction, message distribution, network topology and accuracy and precision

1. What synchronization method in respect to cost, security and precision is suitable for LED synchronization on the C1410 speaker?
2. Will synchronization of LEDs on the speakers expand the possible area of use in respect to the built-in components (PIR sensors and LEDs)?

1.4 Limitations

Given that we are writing the thesis in connection with the development of functionality for Axis' given network speakers, the project will be limited as follows:

- The language in which the prototype is written will be limited to C.
- All synchronization methods will be limited to those that are suitable for an IP network.
- The prototype will be specifically developed for Axis network speakers and are not made for general purposes.
- The network speakers run on a lightweight Linux system.
- A limited number of devices to test on.

2 Technical background

The following chapter will aim to introduce and explain concepts and relevant work essential to understand the experimental results and the technical implementation of the prototype developed.

2.1 Time

The internal unit for timekeeping in a computer, i.e., the real-time clock (RTC), is based upon a crystal oscillator that oscillates with a set frequency, generally, 32768 HZ (215), when voltage is applied [5] [6]. The piezoelectric properties of these crystals, primarily quarts, converts the mechanical work to an electrical charge, which in turn generates a pulse with the resonant frequency of the crystal. Furthermore, as the crystal characteristic frequency is a power of 2, it enables a MOD-65536 counter, also referred to as a 16-bit binary counter, to perform frequency division down to 1 HZ (20). The pulse generated from the counter can thus control the system clock [5].

2.1.1 Clock Offset

The *clock offset* can be defined as the difference between the local clock on the machine and the Coordinated Universal Time (UTC), i.e., true-time [7]. The equation Eq. (1). validates, however, that clock offset can be measured against any arbitrary true-time. Consequently, UTC can be replaced with any clock C1 on the network, thus enabling measuring offset of another clock C2 on the same network relative to C1 [7]. Furthermore, IETF RFC2330 defines clock accuracy as the absolute value of the offset Eq. (2)., where a value closer to zero corresponds to a greater accuracy [7].

$$T_{OFF} = T_{LC} - T_{TTC} \tag{1}$$

$$C_{Accuracy} = |T_{OFF}| \tag{2}$$

2.1.2 Clock Skew

IETF RFC2330 defines the *clock skew* as the frequency difference between the local clock and true-time Eq. (3)., i.e., the first derivative of offset with respect to true-time [7].

$$Skew = f_{LC} - f_{TTC} \tag{3}$$

2.1.3 Clock Drift

The clock skew mentioned in the previous section has a strong correlation with clock drift. The reason that clock drift occurs can be traced back to the frequency of the crystal oscillating changes over time. This loss in frequency accuracy can either be the result of environmental changes, such as temperature and pressure or simply deterioration due to the spanning of time [6]. As described by IETF RFC2330, all clocks experience some variation of skew due to clock drift [7]. Elimination of clock drift would thus result in a constant skew, as seen in Eq. (3).

2.2 Time Synchronization Protocols

As explained in chapter 2.1 all clocks experience some drift. Consequently, a solution for this is necessary for time critical applications. The following two subchapters will explore and explain two protocols used for time synchronization, more precisely, NTP and PTP.

2.2.1 NTP

The Network Time Protocol, from here onwards referred to as NTP, is a well-used protocol to synchronize clocks over the internet. NTP is considered a master-slave network, see Figure 1, where the hierarchical levels are named stratum. The lower the stratum level for a device, the closer to true-time. A device of stratum 0 can thus be seen as the reference clock of all devices in the hierarchy, i.e., true-time. Right under the reference clock, at stratum 1, we find the primary servers that directly synchronize with the reference clock. Subsequently down the hierarchy, the stratum, which also could be considered the distance from the reference clock, increases with 1. With every passing of a stratum, the device clock accuracy deteriorates until reaching the lowest level devices, called the clients. Furthermore, each stratum between the primary server and client is known as secondary servers. Finally, a system clock in NTP is considered to be synchronized to UTC when its deviations have been reduced below nominal tolerances [8].

NTP uses a variant of the Bellman-Ford distributed routing algorithm to determine the subnet topology based on the shortest path rooted on the primary servers [8]. This design allows the

algorithm to automatically reorganize subnets, to produce the most accurate and reliable time, even when the timing network has a fault [8].

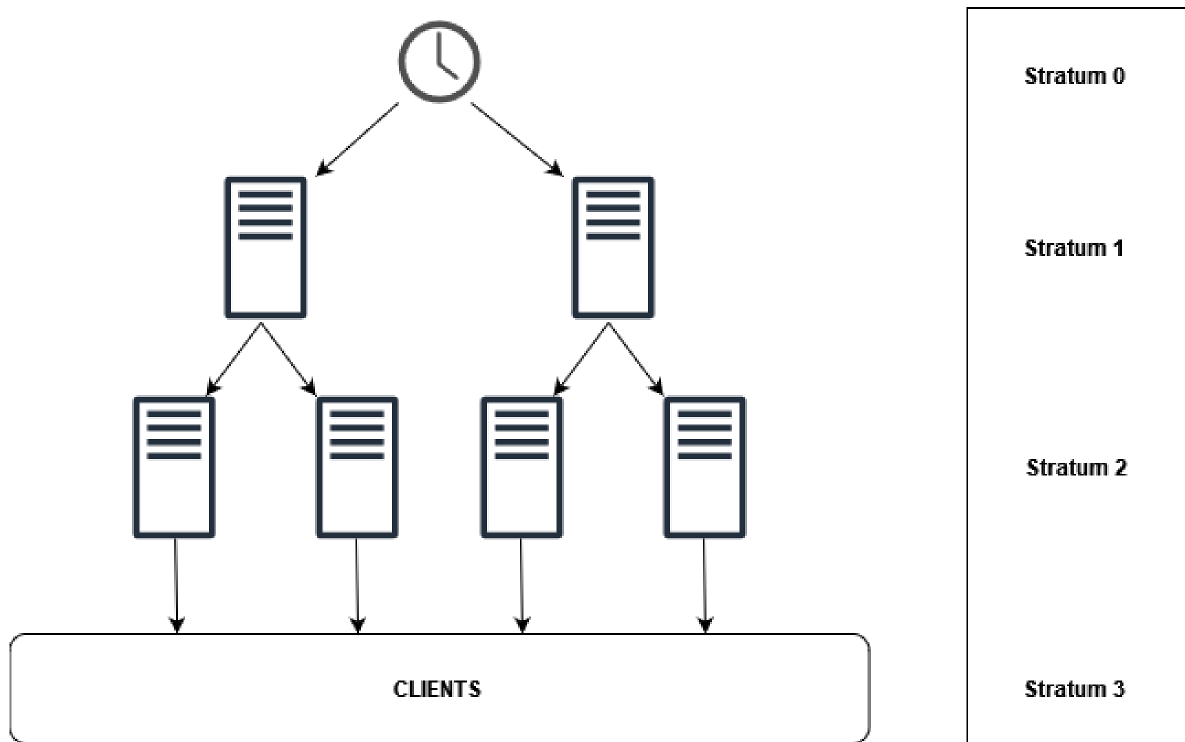


Figure 1: The master-slave hierarchy in NTP with the reference clock at the top at stratum 0.

Clients are synchronized with the primary server by sending packets that contain timestamps. Depending on whether the server or client sent the package, a timestamp can be the UTC date, or the time offset from UTC at time t, respectively. Consider T(t) as the time offset, R(t) as the frequency offset, and D(t) as the clock frequency drift rate. Therefore, if T(t_0) is the UTC time offset determined at some arbitrary time t_0, the UTC time offset at time t can be calculated as seen in Eq. (4), where e is a stochastic error term [8].

$$T(t) = T(t_0) + R(t_0)\,(t - t_0) + 1/2 * D(t_0)\,(t - t_0)^2 + e \tag{4}$$

NTP algorithms are designed to minimize both time and frequency differences between UTC and the system clock. To do this, four different statistics are collected each time it makes a request to a server. The statistics collected are the time offset of the server clock relative to the system clock, round-trip delay between the client and server, dispersion, and jitter, i.e., the root-mean-square average, which is the variance of the recent offset measurements. While

all these statistics are tied to a single server, it should be noted that the NTP protocol includes functionality to combine the statistics of many different servers to synchronize the system clock more accurately [8].

The synchronization and collection of the different statistics previously mentioned for a client with an NTP server is done over UDP. The collection of statistics is based upon requests and replies over the network as seen in Figure 2 [9]. The round-trip delay RTD can be calculated, according to Eq. 5, by ensuring that every package i sent is timestamped T_i, as well as that all previous timestamps T_1 to T_{i-1}, are copied over to the package. Furthermore, the timestamps are used to calculate the time offset according to Eq. 6 [8].

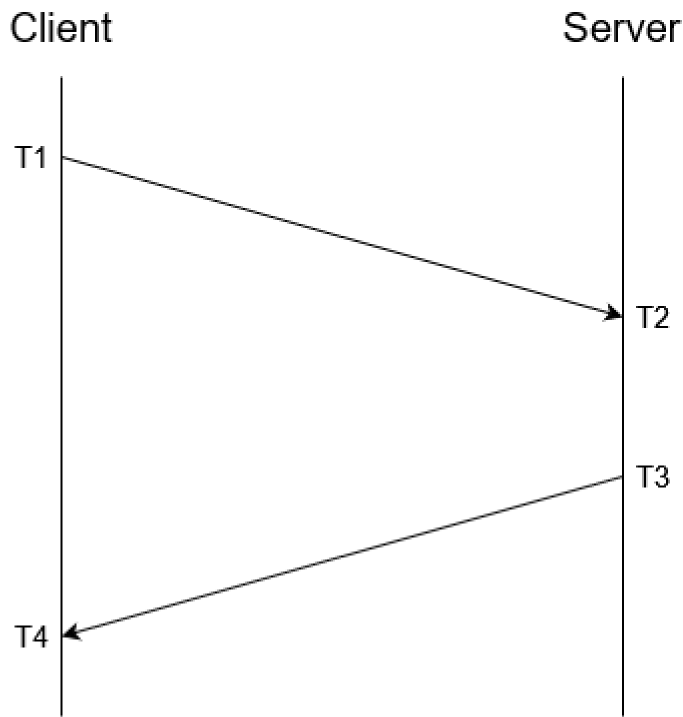


Figure 2: Request and reply over UDP when synchronizing a client with an NTP server.

$$RTD = (T_4 - T_1) - (T_3 - T_2) \quad (5)$$

$$Offset = \frac{1}{2} * ((T_2 - T_1) + (T_3 - T_4)) \quad (6)$$

2.2.2 PTP

The Precision Time Protocol [10], from here onwards referred to as PTP, was primarily written for devices that are used on a local area networks and not wide area networks such as the Internet. Although the protocol can be implemented in such a way that it would work over the internet, it would do so at a diminished capacity, because PTP requires network interface cards that provide hardware support for PTP and network devices such as switches and routers seldom offer such support over the internet. But because the protocol was designed for industrial control, measurement devices, and automation applications in mind, it is seldom a problem that the protocol wasn't designed for a wide area network.

When discussing clocks, it is helpful to introduce three terms that are frequently used in IEEE Std 1588-2019. The first type of clock is an ordinary clock, which usually refers to a clock that belongs to a regular terminal device that has one port. Clocks that are network devices, such as switches that have many ports, are usually referred to as boundary clocks. Devices that measure and communicate latencies are referred to as transparent clocks [10].

Much like NTP, PTP also uses a master-slave synchronization hierarchy, albeit more focused on a LAN setting as opposed to WAN, as seen in figure 3. The hub that is labeled 0 is known as the grandmaster clock, all clocks ultimately derive their time from this clock. This is done by having each level of the tree synchronize their clocks to match their master's clock one level at a time. Each leaf node is known as an ordinary clock, and as can be seen in figure 3 below, the leaf nodes consist of all the terminals and hub2. All the hubs that exist as an intermediate step in the tree are known as boundary clocks, and as can be seen in figure 3, the boundary clocks consist of hub1, 3, and 4 [10].

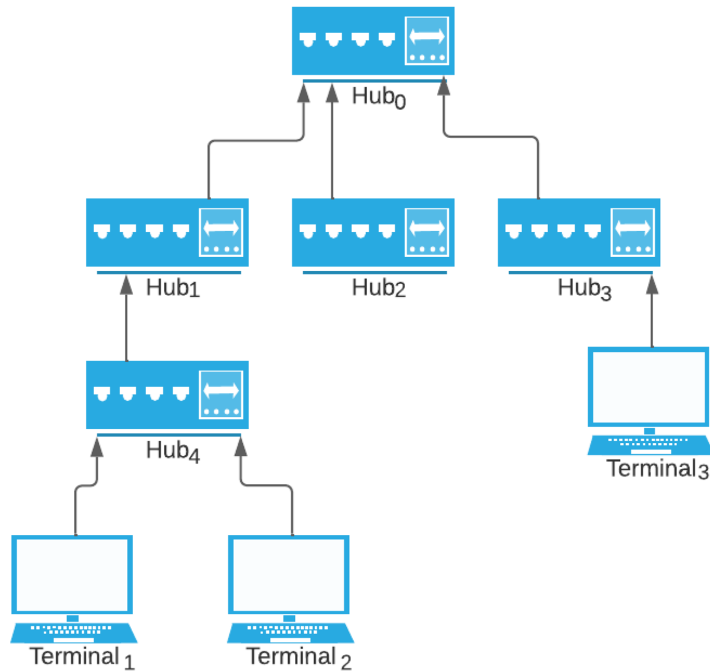


Figure 3: PTP master-slave synchronization hierarchy.

All synchronization takes place at adjacent clocks in the hierarchy i.e., in figure 3 terminals 1 and 2 are only synchronized with hub 4 and hub 4 is only synchronized with hub 1 and hub 1 is only synchronized with hub 0, the grandmaster clock. To make sure that all clocks in the network are synchronized with the grandmaster it is important to make sure that there are no cyclical relationships in the network topology [10]. The physical world is however not always this straightforward, as switches may interconnect in a mesh and form cyclical relationships as seen in figure 4. If PTP is to work on top of such a physical topology, we have to either use a protocol that hides the mesh topology or turn some boundary clock ports passive, which will cause the link to stop working resulting in a tree topology, which the master-slave hierarchy requires. The hierarchy is further determined by putting ports into different states, such as the master or slave state [10].

To determine where all the clocks should be in the hierarchy, PTP requires a state machine to be implemented. Each port of each clock in the network will execute a copy of this state machine. The state machine has several states it can be in, such as the calibrated state, pre-master state, disabled state, or as previously mentioned, the passive state. For the sake of simplification, a network limited to a single network segment with only three states will be

discussed in this section. The states to be discussed are the master state, the slave state, and the listening state. At the startup of a machine, each port that is attached to a network segment of which other PTP ports are attached, will be initialized in the listening state. In this state, the port will wait for a message in the form of an announcement, which is a periodic PTP message that is only transmitted by PTP master ports. Boundary clocks never forward these announcements, so each port can only receive announcements on the network segment it is attached to. The listening state exists to check if there already is a master port in the network segment to which the port is attached. The listening state is limited to a set amount of time before it timeouts. It will assume there is no master port in its network segment if it did not receive an announcement before it timed out and will become the master if it did not receive an announcement before it timed out. The port will now begin transmitting periodic announcements to its network segment once it assumes the master state. The new clock compares itself with the existing master by using the information sent by the announcement messages if a master port was already available when the new clock was in the listening state. So, if the new master's clock is better than the old master's, the old master will receive announcements from the new master. As a result, the old master will assume the slave state [10].

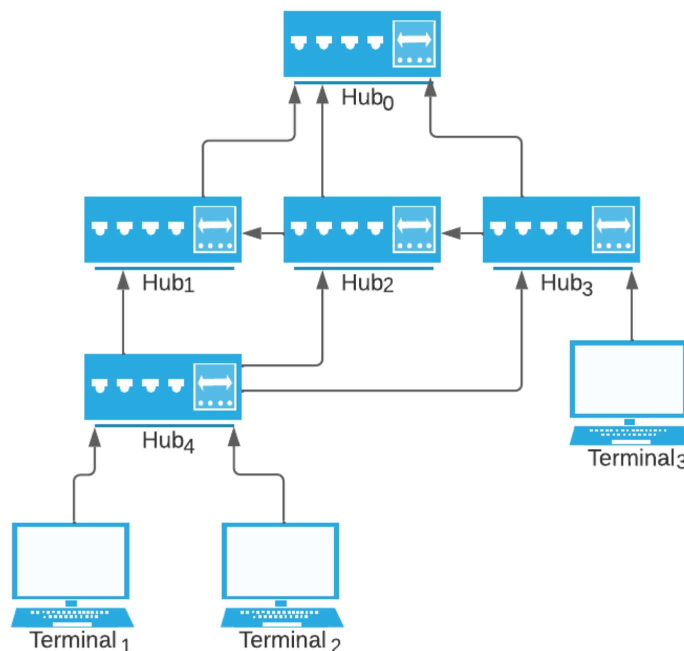


Figure 4: A mesh topology where several boundary clocks are interconnected.

PTP cannot be used on a physical topology like this unless we use a protocol that hides the mesh topology or we put some connections of the boundary clocks into passive mode, triggering the deactivation of certain links and the implementation of the master-slave hierarchical structure. The hierarchy is further determined by putting ports into different states, such as the master or slave state [10].

2.3 MQTT

MQTT is a publish and subscribe protocol for messaging that runs on top of TCP. Running on top of TCP gives MQTT the inherited characteristics of reliable and bidirectional connections. Provided by MQTT are decoupling of the subscribing clients from the publisher, a message transport that is data agnostic, i.e., the payload format is irrelevant, three different services for message delivery, a small transport overhead, and a mechanism for recovering from connection loss [11 p. 1-2].

2.3.1 The Broker

In MQTT, both subscribers and publishers, are considered to be clients and to decouple these, a second party is introduced, the MQTT broker. The broker acts as a server in the middle to which clients can publish and subscribe to so-called topics [12]. After a topic is published, the message is forwarded by the broker to all clients who subscribed to that topic [13].

2.3.2 Broker-Client Connection

When a client connects to a broker, a CONNECT message is sent to the broker containing the mandatory unique clientID and six other optional fields depending on the connect flag bits set, as seen in Figure 5 [11 p. 30-31]. After receiving the CONNECT message, the broker responds with a CONNACK message containing a session present flag and a Connect Reason Code, where 0x00 indicates that the connection is accepted [11 p. 44-46]. After the client broker connection has been established the client can subscribe to and publish data on the information channel corresponding to the topic's name of choice [13].

Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain Flag	Will QoS Flag		Will Flag	Clean Start Flag	Reserved

Figure 5: Bit flags to specify present optional fields in the CONNECT message.

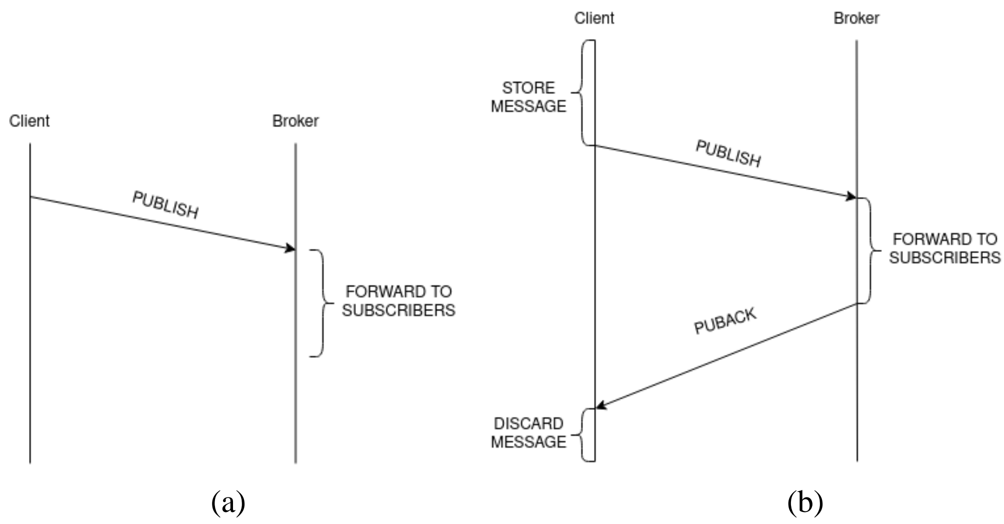
2.3.3 Publish And Subscribe

A client publishes a message by sending a so-called PUBLISH message to the broker on a specific information channel identified by the topic name. The broker is then tasked with forwarding this message to all parties that have subscribed to that channel via the SUBSCRIBE message [13]. The PUBLISH Fixed Header as seen in Figure 6, contains one configurable field, the Quality-of-Service level (QoS) and two flags, DUP and RETAIN [11, p. 53].

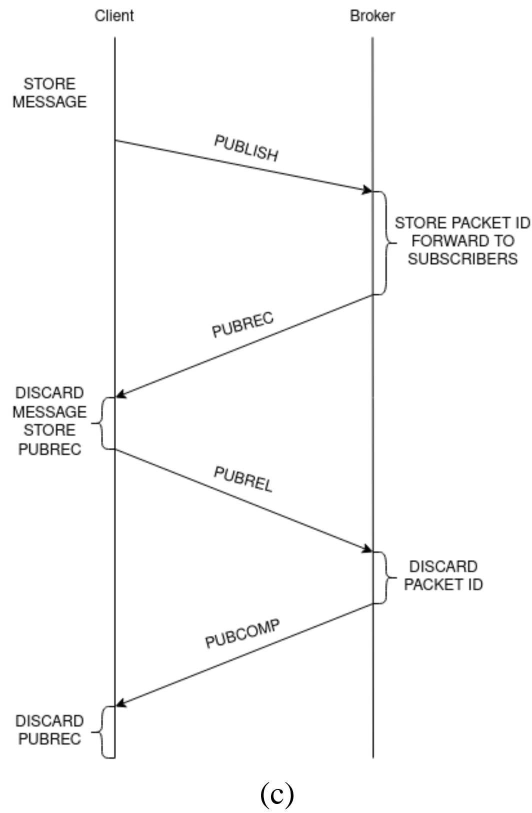
Bit	7	6	5	4	3	2	1	0
	MQTT Control Packet Type				DUP FLAG	QoS Level		RETAIN

Figure 6: The PUBLISHED fixed header. Bit 7 - 4 distinguishes the packet type, 0011 (3) for a PUBLISH message.

Firstly, a QoS level 0 indicates that delivery solely relies on TCP and does not guarantee that the message is processed properly and forwarded to the subscribed clients [11, p. 94]. Additionally, a QoS level of 1 guarantees that the message is delivered to the subscribers at least once. This is accomplished by the sender saving the message until receiving a PUBACK from the broker, indicating that the message has been processed correctly [11, p.94-95]. Finally, the last QoS level, 3, guarantees that the message is delivered once and only once, i.e., no duplicate messages will be processed.



(a) (b)



(c)

Figure 7: (a) Transmission of a PUBLISH message using QoS 0. (b) Transmission of a PUBLISH message using QoS 1. (c) Transmission of a PUBLISH message using QoS 2.

A client subscribes to a topic by sending a SUBSCRIBE message to the broker. The SUBSCRIBE message payload includes one or more topics from which the client requests to receive application data. Upon receiving the SUBSCRIBE message, the broker acknowledges the subscription with a SUBACK message to the client [11, p.71-76].

2.3.4 Topics

The topic name is a UTF-8 encoded string that identifies the information channel for a published message [11, p.55]. The topic name itself can be divided into topic levels where each level is separated by a forward slash. This hierarchical property of topic levels enables a client to subscribe to multiple children of a topic level, either by the single-level wildcard or the multi-level wildcard. Therefore, the single-level wildcard can be considered an arbitrary placeholder representing that topic level, whereas the multi-level wildcard corresponds to multiple sub-levels [11, p.98-99].

(a)	layout/devices/<deviceID>/brightness
(b)	layout/devices/<deviceID>/position
(c)	layout/devices/+/brightness
(d)	layout/devices/#

Figure 8: (a) and (b) is a topic where the forward slash divides the topic in levels where layout is the root. (c) is an example of a use case of a single-level wildcard. (d) is an example of a use case of multi-level wildcard.

2.4 Build system

When deploying software, the build process can be automated by using a build system. This subchapter will discuss two build systems, Bitbake and Meson.

2.4.1 Bitbake

Bitbake is a task execution system that helps to automate the process of assembling entire embedded Linux distributions [15]. Bitbake recipes are files that contain metadata that is

needed to assemble software. These recipe files provide Bitbake with both build and runtime dependencies such as but not limited to, where the source code is and how to get it, information about the hardware and the distribution it is about to run on [17]. The result of the Bitbake assembly process is an image. Images are the binary output that runs on specific hardware.

2.4.2 Meson

Meson is a build system for automating the build process. The Meson build system uses a two-step process to generate so-called build targets, for example, executables or libraries. The first step in the process is the configuration step which includes but is not limited to checking for dependencies, which compiler to use, and setup of unit tests to be run in the build. The output of this first step is the build system, the second and final step consists of executing the build system to generate the build targets [14].

2.5 JSON

JavaScript Object Notation or JSON is a text format that enables serialization of data, i.e., that the objects or data structures can be represented by text, and either be saved to memory or sent over the network for later recreation [18]. Furthermore, JSON is language independent [19] and has four primitive types: strings, numbers, Booleans and null and supports two data structures: objects and arrays [18].

2.5.1 JSON Object

A JSON object seen in Figure 9, consists of name and value pairs where the name is a string and functions as a key to retrieve the value. The value can be any of the previously mentioned primitive data types that JSON supports or another object or array [18].

```json
{
    "Person" : {
        "name" : "Kenny",
        "lastname": "Starfighter",
        "age": 300,
        "phone": {
            "home": "040-000000",
            "ofice": "070-000000"
        }
    }
}
```

Figure 9: Example of a JSON object "Person" with the keys: name, lastname, age and phone. The key phone represents a nested object with the keys: home and office.

2.5.2 JSON Array

A JSON array may range from zero to more values and allows for different types and data structures in the same array. For example, a single array may consist of numbers, strings, and objects [18].

2.6 API

API stand for Application Programming Interface, APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols.

2.6.1 Apache HTTP Server

Apache [23] allows communication between clients and servers over networks by using the TCP/IP protocol. Apache can handle a wide variety of protocols, but HTTP/S is the most common. HTTP/S stands for Hyper Text Transfer Protocol (secure) and is one of the most popular protocols on the web. HTTP/S defines how messages are formatted and transmitted across the internet and how browsers and servers should respond to requests and commands. HTTP Secure is usually accessed via port 443 and HTTP Unsecured is accessed via port 80. Using config files, Apache servers are configured with modules that control their behavior. Apache listens by default to the IP addresses configured in its configuration files.

Listen directives allow Apache to accept and route requests to specific ports and domains based on specific address-port combination requests. Listen runs on port 80 by default, but Apache can be bound to different ports for different domains, allowing a single server to host many different websites. The moment a message reaches its destination or recipient, it sends an acknowledgment message, or ACK message, acknowledging that the data has arrived successfully. A destination host or client will send a Not Acknowledged, or NAK, message if there was an error in receiving data or if some packets were lost on the way.

2.6.2 SOAP

The SOAP [24] protocol specifies the way structured information is exchanged when implementing web services in computer networks. Messages are formatted using XML Information Set and are negotiated and transmitted using application layer protocols such as HTTP. Libsoup [25] uses the SOAP protocol to implement a HTTP client /server library which can be used to set up a synchronous callback-based API.

3 Method

Prior to the thesis a method plan was developed that was divided into different phases with a waterfall approach. The first phases included literature studies and interviews to acquire a better understanding of how to solve the problem. The following phases were focused on the development and testing which would result in a high-fidelity prototype. Finally, the last phase was the writing of the report and presentation.

However, the waterfall method approach was abandoned early on as it was discovered that it was easier to work more iteratively with each step in the development, i.e., all phases were included for each stage of the development. For example, when analyzing the synchronization protocols, all the development, testing, and report writing was conducted before moving on to MQTT.

An overview of the development that took place to create a working prototype is seen in figure 10. For the next stage to be completed the previous stage had to be done for the next stage to be useful. For example, if there is no API from which the program can be controlled then there is no need for a GUI.

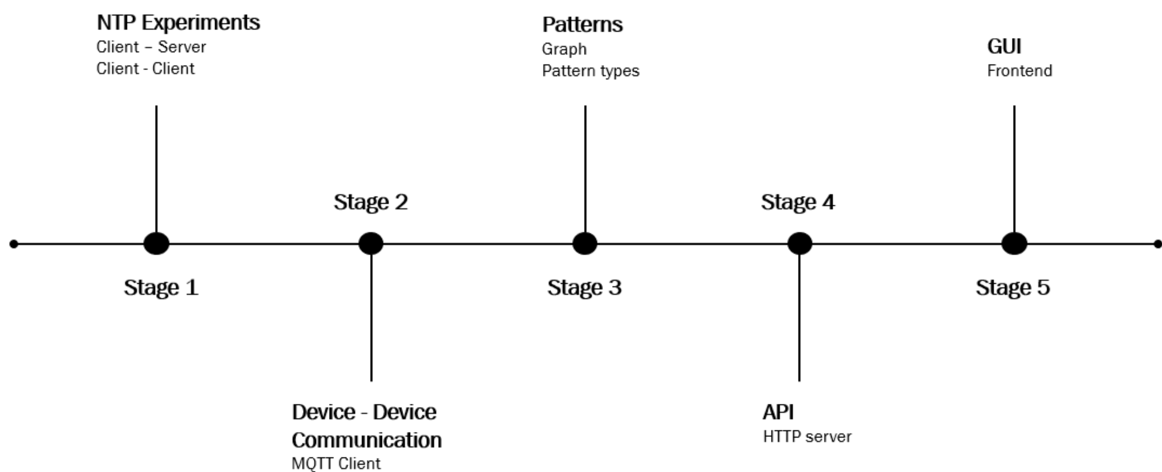


Figure 10: An overview of the development.

3.1 NTP Experiment

To make sure that the network speakers were synchronized we had to investigate the offset that naturally takes place as not all clocks are the same. The purpose of examining the offset of a certain clock has compared to true time, is to make sure the light patterns to be developed aren't noticeable out of sync.

To evaluate NTP, a small program was developed which can be partially seen in Appendix 1 and Appendix 2. The program collects synchronization relevant data, such as offset, delay, jitter, and the next poll against the NTP server. It is worth mentioning that the program is run when using the openNTP's implementation of the protocol. The openNTP implementation was chosen as a starting point because it is what Axis already uses, there are however other implementations. The other implementations were not tested because openNTP was deemed to be good enough for the prototype and other parts of the devices relied upon it. Finally, the raw data from the experiment was imported into MATLAB to visualize the data.

This is the setup of the experiment, a local NTP server setup by axis was used to synchronize.

- **Device:** C1410 (minion)
- **Protocol:** NTP
- **NTP Server Stratum Level:** 5
- **Local NTP Server**: True

The measurements were conducted over a six-day period, within that timespan a total of 828 polls were made. These polls took place at **uneven** intervals because the algorithm derives its next poll value based on how trustworthy it considers its source to be. While the default value of maximum next poll interval is 1024

To get an indication of how large the offset between two audio devices is relative to each other, a second experiment was made. To conduct the experiment a server-client program was written to mimic how the offset is calculated when synchronizing with an NTP server according to Eq. 6. The test ran for 18 hours with the client device sending timestamp requests to the server device with an interval of one request every thirty seconds. Furthermore, the results from the experiments can be seen in figure 15 – 19.

This is the setup for the second experiment, a local NTP server setup by axis was used to synchronize.

- **Device:** 2 x C1410 (minion)
- **Protocol:** NTP
- **NTP Server Stratum Level:** 5
- **Local NTP Server**: True

Based on the data analyzed from the client-server experiment, two main conclusions were established. Firstly, it was concluded that NTP was a sufficiently accurate synchronization protocol, that is, it could be used to synchronize the patterns in the aspect of server-client offset. Secondly, it was decided that another experiment regarding the synchronization of the system clocks between the devices also needed to be conducted. Therefore, a second program was developed to collect data regarding the offset between two devices. As before, the raw data from this second experiment was imported into MATLAB. Subsequently, the data was used to calculate the mean and max offset as well as the frequency difference between the devices' crystals. Finally, the two experiments results were analyzed together, and the decision that NTP would be used as synchronization protocol when moving forward was taken.

The programs developed for the experiments were built using two build systems: Bitbake and, after consultation with the supervisor, Meson. Therefore, to configure the build correctly to be able to deploy the packages on the devices, research about the two build-systems was conducted. Furthermore, example recipes were provided by Axis employees to help with the configuration. The configuration for the Bitbake recipe would, after discussing the matter with Axis employees, only needed to set up once unless major changes to the package occur. Hence, the open embedded build environment would be set up mainly for further development. Consequently, future dependencies and files could be added easier and handled by the Meson build system.

3.2 Device to Device Communication

The next stage of the development concerned how to handle device to device communication. After consultation with the supervisor at Axis, research about MQTT, the protocol in use for

the audio-site package for IoT messaging, was conducted. Subsequently, a client module that was connected to the broker, already created by Axis, was developed. Furthermore, the client was developed with the help of Libmosquitto, the C Mosquitto client library.

3.3 Patterns

With working communication between clients established and a suitable time synchronization protocol chosen, solutions for the LED patterns could be explored.

The approach chosen was that a user should be able to send two different messages that contain configuration information to a device, which in turn will publish the messages to all other devices on the same site via the client that was created in the previous stage. One message contains the graph that has information about how the devices are physically placed in relation to each other, this message is the layout configuration. The other message contains configuration information about the pattern that is about to be played on the graph, this message is the led configuration. The LED configuration contains information about the target device that all other devices should direct their patterns towards as well as colors that should be used in the pattern and the scheduled time for when the pattern should start. There is also pattern-specific information, such as phase, period, or offset, which is used depending on if the user wants the devices to flash toward the target or move in a pulse or a sinus wave towards the targeted device.

Furthermore, manual tests were constructed by hardcoding messages containing configuration information. These tests were used to confirm that the different pattern algorithms could be used on different layout graphs. Consequently, the tests would ensure that any future errors were most likely not caused by errors in the previous stage. After testing that the layout graphs and the different patterns worked the development of the API could start.

3.4 API

The next stage of the development was to make sure the end user of the prototype could control it in a sensible way. To control the program through a graphical user interface, an

application programming interface, henceforth referred to as API, had to be developed. The Apache HTTP Server running on the devices as a reverse proxy server was configured to redirect requests to a SOAP server module. Subsequently, the SOAP server was used to serve the requests from the user.

The API was manually tested by using the curl command via the Linux Debian terminal to test that the API could handle different POST requests to the server. These POST requests were needed for the end user to send new pattern and layout configurations. The GET requests were simply tested by entering the device's IP into a browser and applying the correct end point after the IP, for example https://172.25.14.79/wayfinder. Once it was confirmed that the API worked as intended, we could use this server to serve the GUI, which was going to be constructed in the next stage.

3.5 GUI

The last stage of development was to construct a front end that could communicate with the API that was constructed in the previous stage. The main problem in this stage was to correctly parse the messages that was sent between the browser and the Apache HTTP Server. The parsing of information was made possible by using JavaScript. Once the messages could be parsed correctly it was only a matter of the user being able to see the current configurations the devices had and then being able to change it and send the new settings to the devices. HTML was used to set up different boxes that both the server and the user could interact with.

3.6 Source Criticism

To describe the background of how this thesis project came to be, sources from Axis Communications AB were used to describe the company. The motive behind using these sources is to make it easier for the reader to understand the circumstances of this thesis. The descriptions are informative but a bit exaggerated in terms of sales purposes at times.

To specify PTP and certain characteristics of time as it pertains to synchronization, IEEE was used as a source. IEEE is the acronym for the Institute of Electrical and Electronics Engineers

and since its publications and technology standards are often cited in papers, it is generally considered a trustworthy source. IEEE is a primary source for their publications and technology standards.

RFC was used as a primary source to describe; certain characteristics of time as it pertains to synchronization, NTP, JSON and SOAP. RFC is the acronym for Request for Comments and RFCs are sometimes official documents of internet specifications, communications protocols, and procedures that the IETF sometimes adopts. RFCs that become official documents are peer reviewed by other engineers and are therefore generally trustworthy.

To illustrate how NTP synchronizes clients with servers over UDP, NTP.org was used as a primary source. NTP.org produces the official reference implementation of NTP along with implementation documentation and is therefore trustworthy when it comes to this subject.

To outline how MQTT works, two different sources were used. The first source OASIS is a non-profit organization that makes open-source projects and standardizations. The second source HiveMQ is a company that provides solutions for connecting devices. HiveMQ has made an implementation of the MQTT protocol that they sell. While HiveMQ have some educational materials that explains the general concepts of MQTT, it is a secondary source.

To give the reader an overview of how Apache works, apache.org was used as a source. Apache.org is owned by the Apache Software Foundation (ASF) which is a charitable organization, funded by individual donations and corporate sponsors. Since apache.org is a primary source of information for Apache HTTP servers and it is run by volunteers with no profit motive, it can be considered a trustworthy source.

To recount how the build system that was used worked, two different sources was cited. The first source is mesonbuild.com, which is a primary source of information regarding the Meson Build system. Meson is a software tool for automating the compiling of software. The second source is the yoctoproject.org which is a primary source for both Bitbake and Openembedded. The Yocto Project is a collaborative open-source project that enables the creation of Linux distributions for embedded software that are independent of underlying hardware.

To define the syntax of valid JSON texts and how it can be used, two different sources were used. The first source that was used is RFC and has already been discussed. The second source json.org have educational materials relating to the syntax of JSON and is a secondary source of information regarding JSON.

To explain how the API was implemented, Libsoup.org was used as a source for information regarding how the Libsoup library implemented the SOAP protocol. Libsoup is a community organization and a primary source of information of the Libsoup library.

To describe certain characteristics regarding clocks as it pertains to synchronization, a book called NextGen Network Synchronization was used as source. The author: Dhiman Deb Chowdhury, is a tech industry veteran with 25+ years of experience in data telecommunication industry and can thus be trusted as a secondary source of information.

To conclude that there is no general synchronization strategy, a research paper called "A methodology for Choosing Time Synchronization Strategies for Wireless IoT Networks.", was used as a source. This research paper has been peer reviewed at the university of Madrid, because of this it can be considered a trustworthy source.

To outline the advantages of different IoT messaging protocols available, a research paper called "Messaging Protocols for IoT Systems – A Pragmatic Comparison", was used as a source. The research paper was published, and peer reviewed by MDPI, which is an acronym for Multidisciplinary Digital Publishing Institute. MDPI is an open access scientific journal and can thus be considered a trustworthy source.

To give credence of the ethical aspects brought up in this paper, a research paper called "Openness versus Secrecy in Scientific Research Abstract", was used as a source. The paper was published and peer-reviewed by the American NIH, which is an acronym of The National Institute of Health and can therefore be considered a reliable source.

4 Results

In this chapter the results from two experiments will be presented that will be the basis of decisions regarding the synchronization protocol for the prototype.

4.1 NTP Server Synchronization

Our testing indicates that the offset usually does not exceed 100ms as can be seen in figure 12 below. On average the offset is 9.98ms, this conforms with the protocol specification [8], which states that any offset remaining while using NTP can be measured in the tens of milliseconds.

We also measured the timing of each poll and as can be seen in figure 14. The algorithm often considered the source to be quite trustworthy as the poll interval often peaks at 1500 seconds with an average polling interval of 618 seconds.

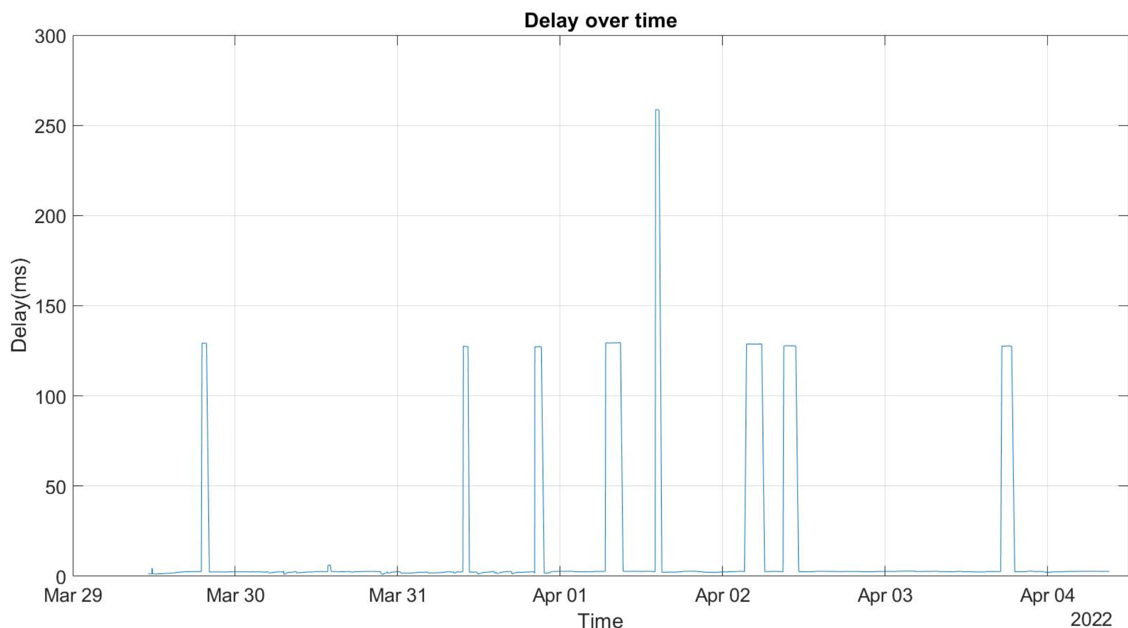


Figure 11: This figure shows the delay over time measured in milliseconds.

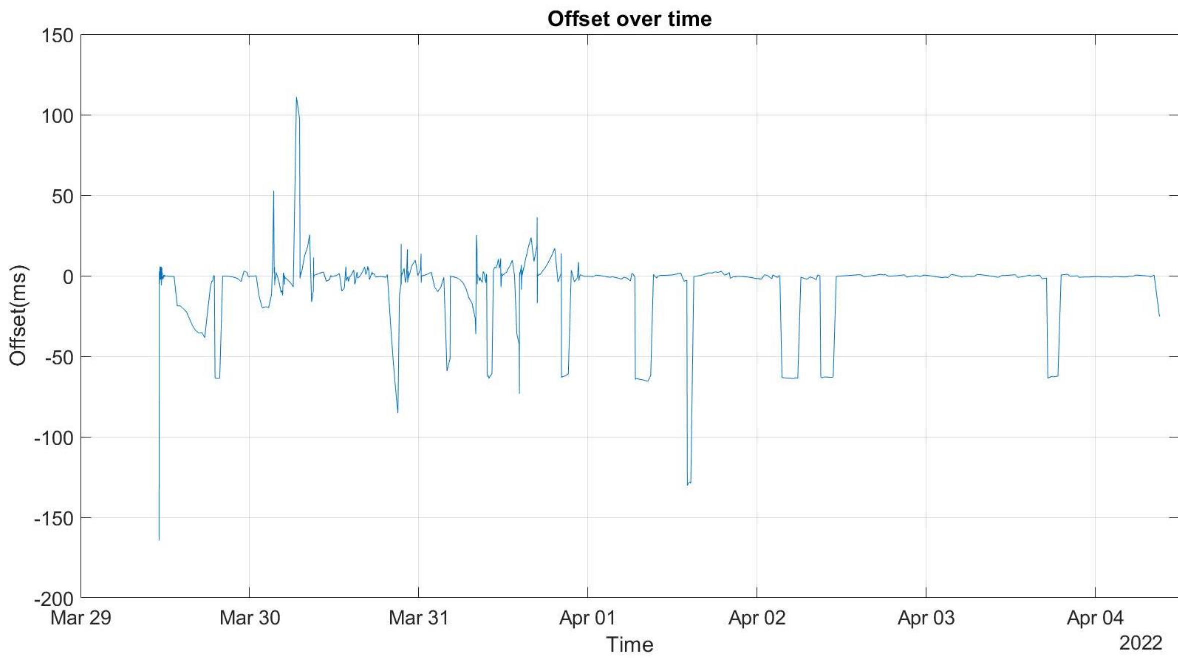


Figure 12: This figure shows the offset over time measured in milliseconds.

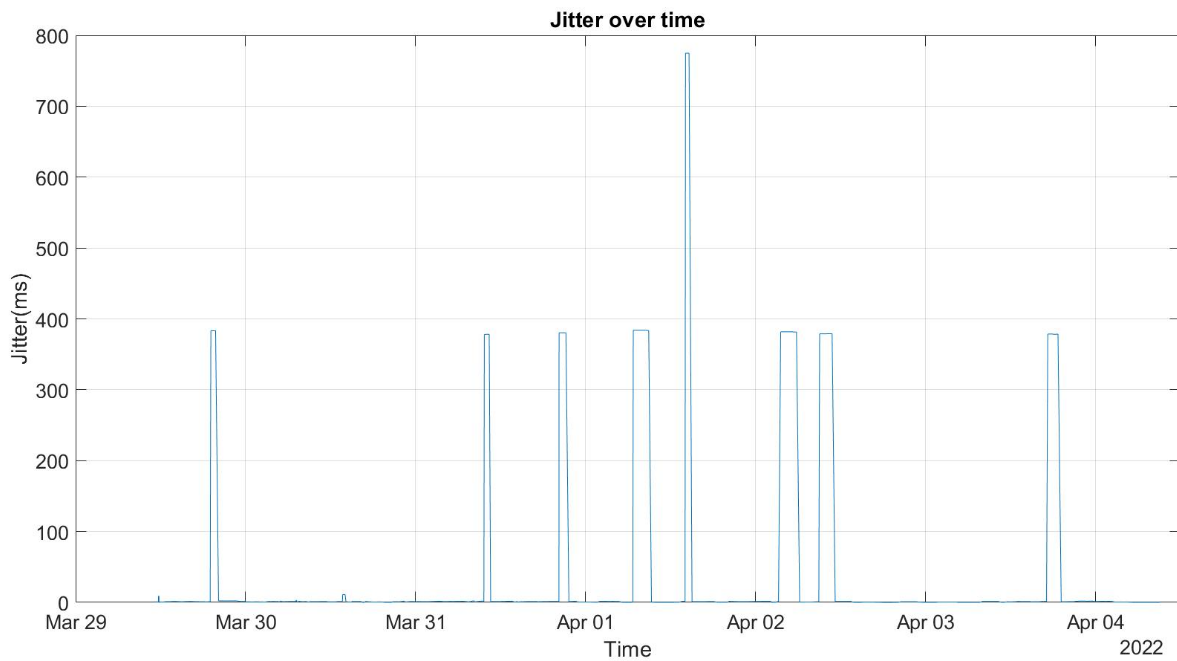


Figure 13: This figure shows the jitter over time measured in milliseconds.

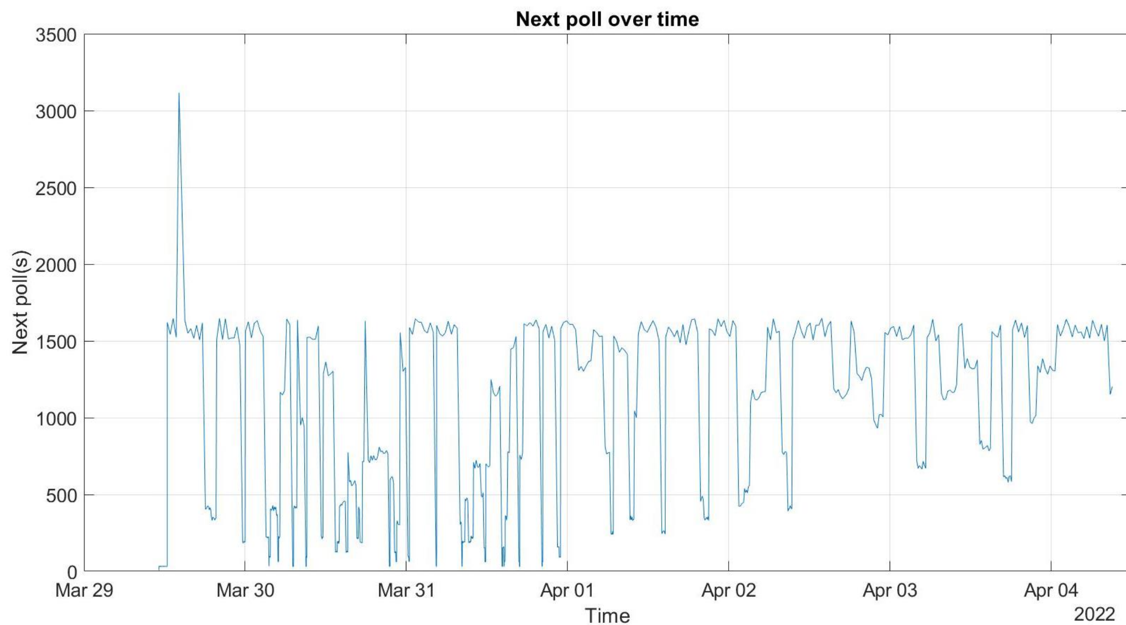


Figure 14: This figure shows the next poll time, over time measured in seconds.

4.2 Time Offset Device – Device

It should be acknowledged that the timestamps for receive time and send time both for server and client are done in the application layer, and no consideration has been taken for processing time nor possible context switches by the OS scheduler. Jitter might also play a role in the offset.

Mean Offset	-1213.49µs
Absolute Max Diff	5319µs
Standard Deviation	1631.83µs
50 percentile	-1079µs
75 percentile	-7µs
90 percentile	427µs
95 percentile	1381µs
99 percentile	2110µs

Figure 15: The mean offset between two audio devices, the absolute value of the maximum difference between two devices, and the standard deviation of the observed dataset.

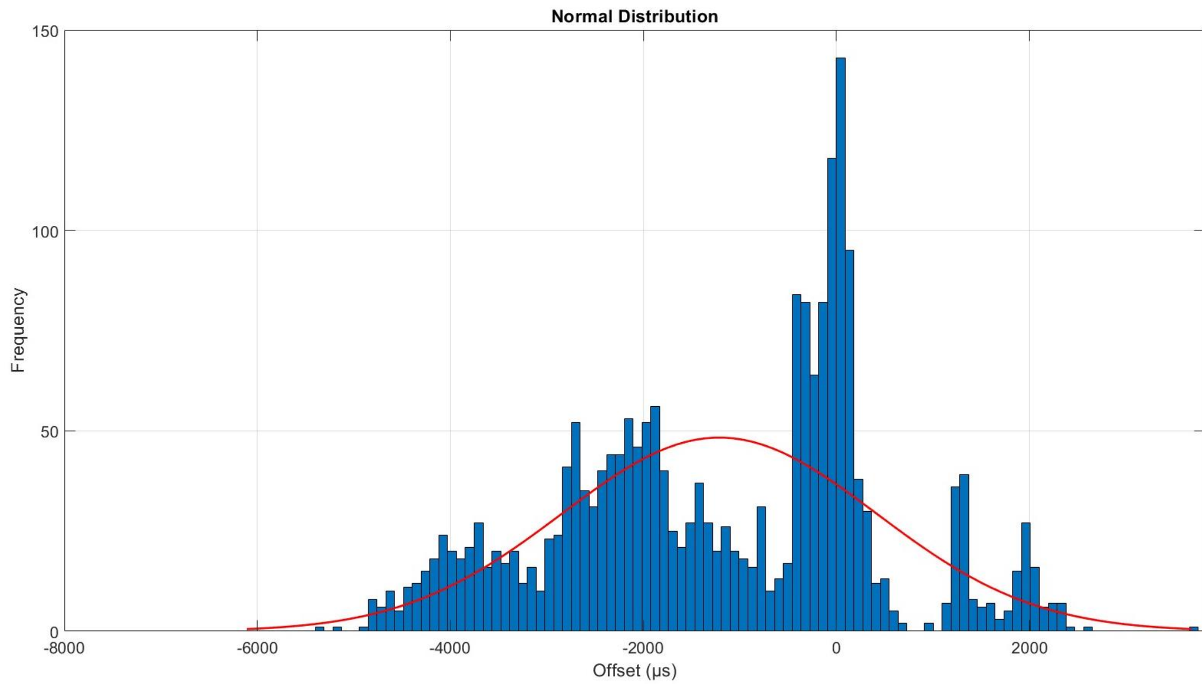


Figure 16: The normal distribution curve of the dataset

Mean Freq Diff	0.0513µHz
Max Diff	102.93µHz
Min Diff	-112.43µHz
Standard Deviation	11.058µHz
Abs 50 Percentile	0.1333µHz
Abs 75 Percentile	3.3000µHz
Abs 90 Percentile	9.6667µHz
Abs 95 Percentile	14.6000µHz
Abs 99 Percentile	28.5000µHz

Figure 17: The mean frequency difference between two devices, the max and min frequency difference, and the standard deviation of the observed dataset.

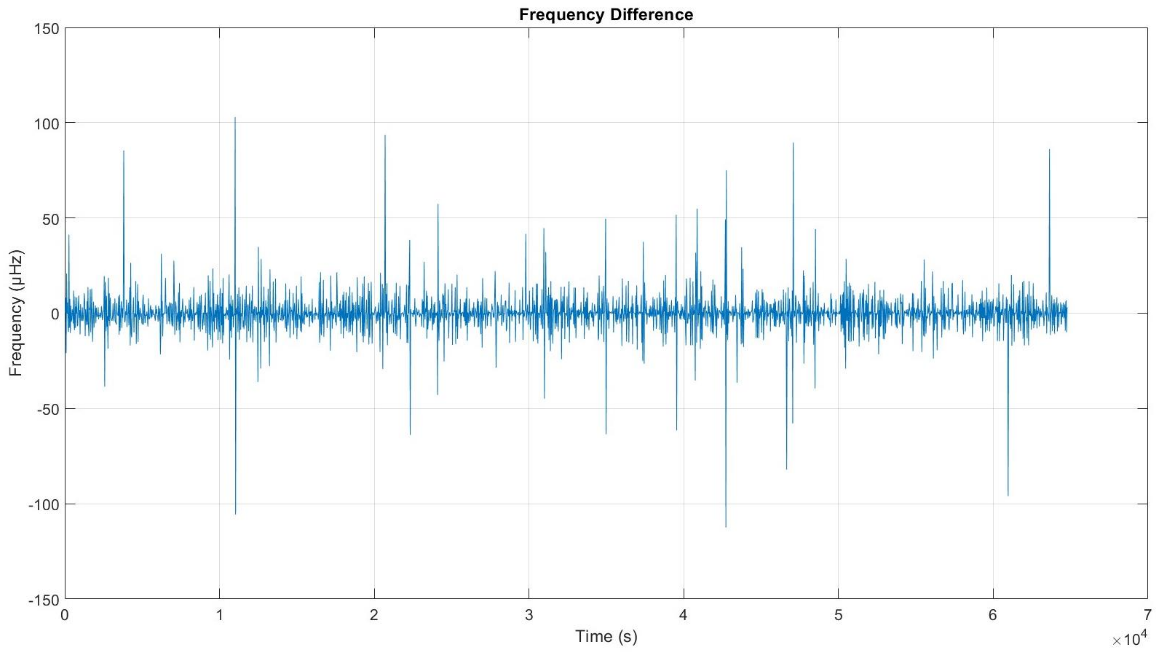


Figure 18: Frequency difference over time (First derivative of the time offset according to Eq. 4). Note that the frequency is displayed in μHz.

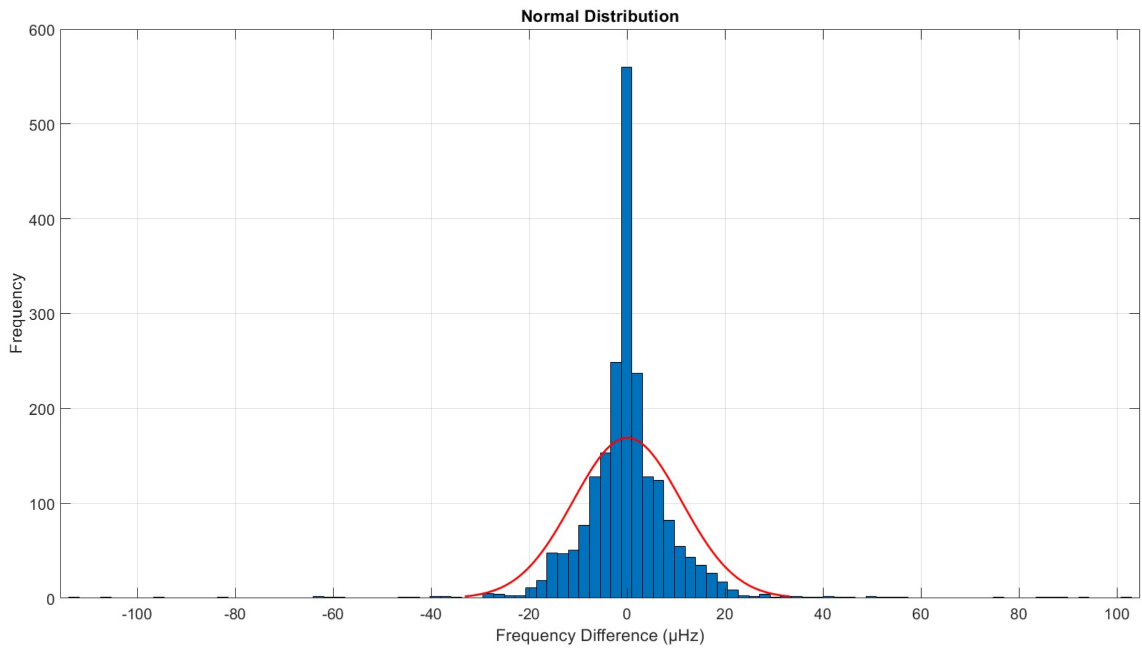


Figure 19: The normal distribution of the frequency difference over time. Note that the frequency difference is displayed in μHz.

5 Analysis

In this subchapter the result of the experiments conducted in chapter 4, will be analyzed as well as all other choices that has been taken during the development.

5.1 Selecting Time Synchronization Protocol

When choosing the synchronization protocol different aspects need to be considered. First, we need to establish what is good enough precision, i.e., how much offset is acceptable between the different devices' internal clocks as well as their offset to true-time. Secondly, the cost aspect of both NTP and PTP needs to be taken into consideration. Finally, security concerns for the chosen protocol needs to be addressed.

5.1.1 Precision

To ease the discussion about precision, we introduce a simple scenario 1a.

Scenario 1a.

1. The user schedules a light pattern at 14:00:00

For scenario 1, the devices should be close to true-time so that the pattern is started at the expected time. According to [8], the precision between client and server has been reported in tens of milliseconds for NTP. However, in the OpenNTP experiment, some peaks, where the offset exceeded more than 100ms, were detected. This is likely a result of the device clock being allowed to drift longer between the polls and it might be solved by another implementation of NTP. Furthermore, the milliseconds precision of NTP can be put in perspective to PTP, where the offset of a pure PTP software implementation and PTP with hardware support has been reported in microseconds and nanoseconds respectively [20]. Nevertheless, a 100ms delay from a scheduled time is most likely not noticeable. Another aspect, more importantly for our led pattern synchronization, is the perceived time of the devices relative to each other as well as the relative clock skew.

The results in chapter 4 indicates that the offset of the system clock between two devices with the same hardware is less than that of a device relative to the OpenNTP server. A delay of approximately five milliseconds, i.e., the max difference from the dataset between LEDs

starting, could most likely be considered neglectable, that is, in the aspect of what a human would be able to perceive. Furthermore, no abnormalities from this assumption have so far been observed when scheduling a pattern using NTP for time synchronization.

Furthermore, regarding the clock skew, a minimum frequency difference of -112.43 µHz and a maximum frequency difference of 102.93 µHz was derived from the offset values following Eq 4. The relative clock skew is important from a synchronization perspective, i.e., the devices should perceive the length of a second as accurate as possible. Based on the process of timestamping in computers, with frequency division mentioned in the technical background chapter, Eq 7 and Eq 8 can be derived.

$$PS = \frac{f}{2^{15}} \tag{7}$$

$$\Delta S = \left(\frac{f_1}{2^{15}}\right) - \left(\frac{f_2}{2^{15}}\right) \tag{8}$$

Therefore, a frequency of 32768 Hz would, according to Eq 7, be considered a perfect perceived second. However, as stated in the technical background, the oscillation of the crystal changes over time. Hence, the perceived second, PS, would differ from one. Consequently, the difference between the perception of a second, ΔS, between two devices could then be calculated following Eq 8. Furthermore, assuming a steady resonant frequency for f_1 in Eq 8, the equation could be rewritten as Eq 9.

$$\Delta S = 1 - \left(\frac{2^{15} - \Delta f}{2^{15}}\right) = \frac{\Delta f}{2^{15}} \tag{9}$$

Based on Eq 9, the maximum perceived differences of a second between the devices is the result of the min and max frequency difference as the input, which can be seen in figure 20.

Δf	ΔS
-112.43 µHz	-3.4310 ns
102.93 µHz	3.1412 ns

Figure 20: The difference between how two devices perceive a second according to Eq 9 with input derived from the experiment in chapter 4.

A difference between LEDs on and off state in nanoseconds is likely to be recognized as imperceptible. Considering the perspective of the thread suspension once the LED have changed state, the OS scheduler's non-guarantee for longer suspension is of greater concern.

5.1.2 Cost

After discussion and examining the existing codebase for the firmware it was concluded that the protocol used today for the speakers, regarding time synchronization, was the NTP protocol. More precisely, the operating system program that handles time synchronization for the speakers is the ntpd implementation. PTP for time synchronization would thus require further development and possible investment in new hardware to fully make use of the precision gain of PTP.

5.1.3 Verdict

The results from the experiment confirmed that NTP was a suitable synchronization protocol for the prototype. Subsequently, the precision errors found were concluded to be neglectable for the use case. However, it should be noted that PTP theoretically exceeds NTP concerning the precision aspect. Therefore, PTP could be a candidate if further research concludes that increase in precision is needed. Where NTP exceeds PTP is that it is more cost efficient, that is, no more development is needed since configuration is already done for the protocol.

According to RFC5906 [26], a time synchronization service should be reliable. Thus, the client should be able to authenticate that the package was sent by the server. The synchronization scheme should in other words be resistant to man in the middle-attacks. NTP supports configuration of Autokey which uses digital signatures and public certificates for authentication [26]. Furthermore, Network Time Security (NTS) has in RFC8915 [27] been specified as an option to provide security for NTP using Transport Layer Security (TLS). The relatively new development of NTS indicates that the NTP security concerns are addressed and future development in the area could be expected to meet future threats.

5.2 Selecting IoT Messaging Protocol

MQTT was chosen as the messaging protocol for our prototype. However, it should be acknowledged that other protocols could have been considered and researched as well. The list of other IoT messaging protocols is long, such as but not limited to AMQP, WAMP, and CoAP depending on the needs [21]. Furthermore, MQTT and CoAP are described as having the advantage of low delivery delays [21], which from a synchronization perspective is seen as an important requirement. Nevertheless, the main arguments for the choice of MQTT as the messaging protocol were that AXIS used it in the underlying architecture and the scope of this thesis.

Building the module on top of the underlying architecture, that is the audio-site package, meant that we could connect our MQTT client module into the existing running broker used by audio-site. Additionally, it allowed us to make use of the already generated PSK to establish a secure TLS connection. Finally, the use of MQTT would most likely ease the integration of our code into the existing codebase hence reducing development cost.

5.3 Data For Synchronization

Choosing a suitable time synchronization protocol is one part of the equation to solve the synchronization problem. Additionally, to be able to synchronize the devices' LEDs it was concluded that data needs be shared among the devices. Consequently, the devices need to be aware of the other devices running the program that are connected to the same broker. Additionally, they need to be informed about the physical location relative to each other. Other shared data needed includes pattern specific variables, such as flashes per minute for a flash pattern, duration for a pulse pattern, and the period and phase for a sine pattern as well as generic variables. A detailed overview of the different variables can be seen in figure 21.

Variable	Flash Pattern	Sine Pattern	Pulse Pattern
pattern_type	X (string)	X (string)	X (string)
target_id	X (string)	X (string)	X (string)
pattern_running	X (0 or 1)	X (0 or 1)	X (0 or 1)
ext_red	X (string)	X (float)	X (float)
ext_green	X (string)	X (float)	X (float)
ext_blue	X (string)	X (float)	X (float)
scheduled	X (0 or 1)	X (0 or 1)	X (0 or 1)
scheduled_time	X (string)	X (string)	X (string)
offset_on	X (0 or 1)	X (0 or 1)	X (0 or 1)
duration			X (float)
flashes_per_minute	X (integer)		
period		X (float)	
phase		X (float)	

Figure 21: The data variables sent to a device to calculate the led configuration. X denotes that this variable is present for the respective pattern.

5.3.1 Graph data

To solve the problem regarding the awareness of the physically layout of the devices different approaches was discussed. One approach discussed revolved around using sound localization, utilizing the speakers' two-way microphones and apply graph theory to build a representation of the layout. The sound localization aspect was however deemed too complicated for the time constraints. Instead, it was decided that the graph data would be sent in manually to one of the devices and later redistributed to the other devices with the use of MQTT. Subsequently, the graph data could be used to find the shortest distance to any arbitrary device in the graph using Dijkstra's algorithm.

5.3.2 LED data

Being able to synchronize the pattern, the devices need to share the same initial LED data, that is, they need to be aware of the pattern type and its configuration. Furthermore, it is necessary that the data can be manipulated in such a way that it can be used for indicating a path to a target device. Consequently, the solution was to include a flag in the LED payload

if the pattern was to indicate a path to a target device. A value of one for this flag, "offset_on", would trigger calculations in the offset or phase shift needed by using the distance to the target node obtained from the graph. Additionally, a zero would indicate that the pattern would not be offset and thus run-in uniform with the other devices.

5.4 External Communication with a Device

Fetching necessary data about the current layout and active device members for a graphical interface introduced the need for a rest API. Furthermore, the development of a rest API would ease posting new graph data and LED configuration to a device for further redistribution with MQTT. Consequently, this meant that the program would, in addition to running the MQTT client, also have a module acting as an HTTP server for incoming POST and GET requests. To realize this, the Apache httpd running on the devices' acting as a reverse proxy server, needed to be configured so that requests to the endpoints would be proxied to the server [22].

6 Implementation

The result of this thesis is an executable prototype program for the C1410 minion network speaker that uses NTP for time synchronization and can produce three different light patterns: flash, pulse, and sine. After concluding two experiments regarding NTP and researching the alternative PTP, NTP was deemed to work sufficiently enough for our use case.

The devices' LEDs can be uniform i.e., no offset in time from one LED value changing on one device relative to that of one on another device. Additionally, the LEDs can have an offset relative to each other producing a path in the selected light pattern to a target device.

6.1 Device-Device Communication

The program handles device-to-device communication with the IoT messaging protocol MQTT by subscribing and publishing to different so-called topics seen in Figure 22. Additionally, as previously mentioned in the analysis chapter, the connection is secured with the TLS protocol using a pre-shared key established by audio-site.

ID	Topic Name
(a)	wf-audio-light/layout/edges
(b)	wf-audio-light/target
(c)	wf-audio-light/layout/join
(d)	wf-audio-light/layout/disconnect
(e)	wf-audio-light/layout/member/request/<id>

Figure 22: The topics used for messaging between connected clients.

A client subscribes to all topics seen in figure 22 on startup when a connection to the established broker running on the underlying architecture, the audio-site, has been set up. Furthermore, a publish to topic (c) as well as the last will to topic (d), both with a payload containing the device ID, occur when the connection is established. Consequently, this exchange of messages on startup enables the devices to get information about other available devices. Finally, the subscriptions also establish the necessary communication channels for other data exchange between the devices, which, as mentioned in the chapter 5.3, are essential for synchronization. An overview of the data flow on connection can be seen in figure 23,

and the result of this data exchange to the local files on the devices can be seen in figure 24. Additionally, a snippet of the code for the connection can be found in appendix 7.

In contrast to the join topic (c), the disconnect topic (d) is subscribed to so that information about clients disconnecting is received and removed from the member's file. This is achieved by publishing a last will message to the broker containing the device ID, which is retained and redistributed when a connection loss is recorded.

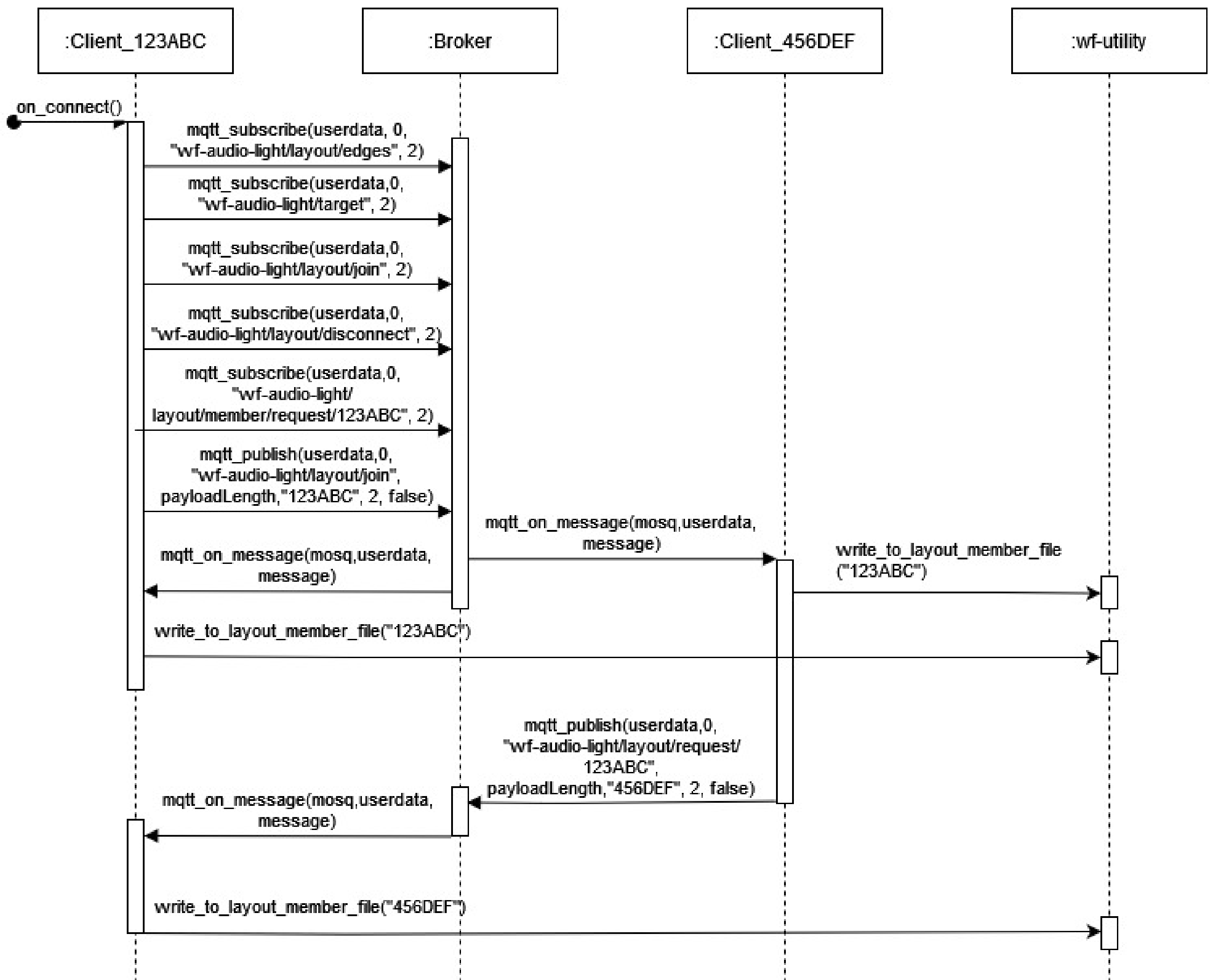


Figure 23: Sequence diagram for when a client with ID "123ABC" have received an CONNACK and established a connection with the broker when one other client with ID "456DEF" is running.

```
{
    "members" : [
        "456DEF",
        "123ABC"
    ]
}
```

Figure 24: The content of the local members file on the devices after message exchange when a new client with ID “123ABC” joins when one other client with ID “456DEF is running.

The distribution of the graph data to the devices are handled by subscription to topic (a). The payload of the MQTT message is in JSON format that on received is stored to a file. Subsequently, this data is used for calculating the distances to the other devices in the graph. An example of the graph payload can be seen in figure 25.

```
{
    "deviceMembers" : [
        {
            "id": "ae08c6b4-c1e9-4c95-ab47-44e977a62a33D",
             "edges": [
                "e93fea84-312f-472d-8292-5ef2f63c2cbfD"
             ]
        },
        {
            "id": "e93fea84-312f-472d-8292-5ef2f63c2cbfD",
            "edges": [
               "ae08c6b4-c1e9-4c95-ab47-44e977a62a33D",
               "04a0d971-a5e5-4722-9847-6b571c1c9e52D"
            ]
        },
        {
            "id": "04a0d971-a5e5-4722-9847-6b571c1c9e52D",
            "edges": [
            "e93fea84-312f-472d-8292-5ef2f63c2cbfD",
               "c88b0864-d556-48d0-9b36-6f0d674ca50cD"
            ]
        },
        {
            "id": "c88b0864-d556-48d0-9b36-6f0d674ca50cD",
            "edges": [
               "04a0d971-a5e5-4722-9847-6b571c1c9e52D"
            ]
        }
    ]
}
```

Figure 25: Example payload for the topic “wf-audio-light/layout/edges”. Each device has a unique ID with the key “id” and can have zero to many edges. Thus, the payload represents a graph of devices with its corresponding edges.

Finally, the last topic of interest (b), the target topic, is the communication channel for pattern data used to configure the LED file. Subsequently, this LED file is later used to control the pattern in aspect of on and off time, and color of the LEDS.

6.2 Graph

The graph represents the physical location of the devices relative to each other. The graph is derived from a local graph file on the devices and changed by publishing to the topic “wf-audio-light/layout/edges” previously stated.

All the edges in the graph have a set weight with the value one. This weight, i.e., cost to an adjacency device, is later used when calculating the distance matrix with Dijkstra’s algorithm. Consequently, this approach results in global information for all distances from an arbitrary device to any other device in the graph for each client. The payload from figure 25 would thus result in the graph seen in figure 26 and in the distance matrix seen in figure 27.

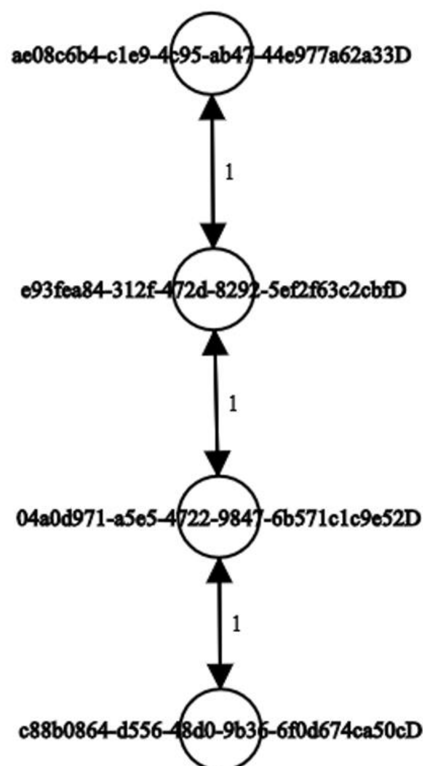


Figure 26: Graphical representation of the payload in figure 25, each edge with a weight of one.

	ae08c6b4-c1e9-4c95-ab47-44e977a62a33D	e93fea84-312f-472d-8292-5ef2f63c2cbfD	04a0d971-a5e5-4722-9847-6b571c1c9e52D	c88b0864-d556-48d0-9b36-6f0d674ca50cD
ae08c6b4-c1e9-4c95-ab47-44e977a62a33D	0	1	2	3
e93fea84-312f-472d-8292-5ef2f63c2cbfD	1	0	1	2
04a0d971-a5e5-4722-9847-6b571c1c9e52D	2	1	0	1
c88b0864-d556-48d0-9b36-6f0d674ca50cD	3	2	1	0

Figure 27: The distance matrix from the graph in figure 25.

The distance matrix is stored in the client, as seen in Appendix 4. Subsequently, the matrix is used for calculating the correct offset, on and off time, or phase shift in response to receiving data from the target topic.

6.3 Patterns

As previously mentioned, the result of the thesis is a prototype that can produce three different so-called patterns: flash, pulse, and sine. Furthermore, the pattern data is distributed to all devices on the “wf-audio-light/target” channel seen in figure 22 after a post to the API endpoint on the server. Furthermore, the LED of a device can be controlled by changing the values of a configuration file, this was already made possible by Axis..

6.3.1 Flash

A POST to start a flash pattern has the structure seen in figure 28 with the pattern specific variable fpm i.e., flashes per minute.

```
{
    "pattern_type" : "flash",
    "target_id" : "ae08c6b4-c1e9-4c95-ab47-44e977a62a33D",
    "pattern_running" : 1,
    "ext_red" : "255",
    "ext_green" : "0",
    "ext_blue" : "0",
    "scheduled" : 1,
    "scheduled_time" : "15:30:00",
    "offset_on" : 1,
    "fpm" : 120
}
```

Figure 28: The structure of the payload when scheduling a flash pattern at 15:30 with full color intensity of red that should indicate a path to the device with ID ae08c6b4-c1e9-4c95-ab47-44e977a62a33D.

The fpm variable has two contexts depending on the "offset_on" flag's value. It can express how many times per minute a LED on any arbitrary device should switch between on and off when the flag has a value of 0. Additionally, for a flag value of 1, as seen in figure 28, the fpm value implies how many times per minute the LED is on considering all the devices together as well as the graph layout. Consequently, the time between the on and off state for the devices depends on the longest path to the target device as visualized in Figure 29. It is worth mentioning that devices with the same distance to the target device are synchronized in the switch between the on and off state. For example, the devices B, C, and D in figure 29, are on or off at the same time. Additionally, only devices that have the same distance to the target can be on at the same time as visualized in figure 30. The code snippet for calculating the on and off times can be seen in Appendix 6.

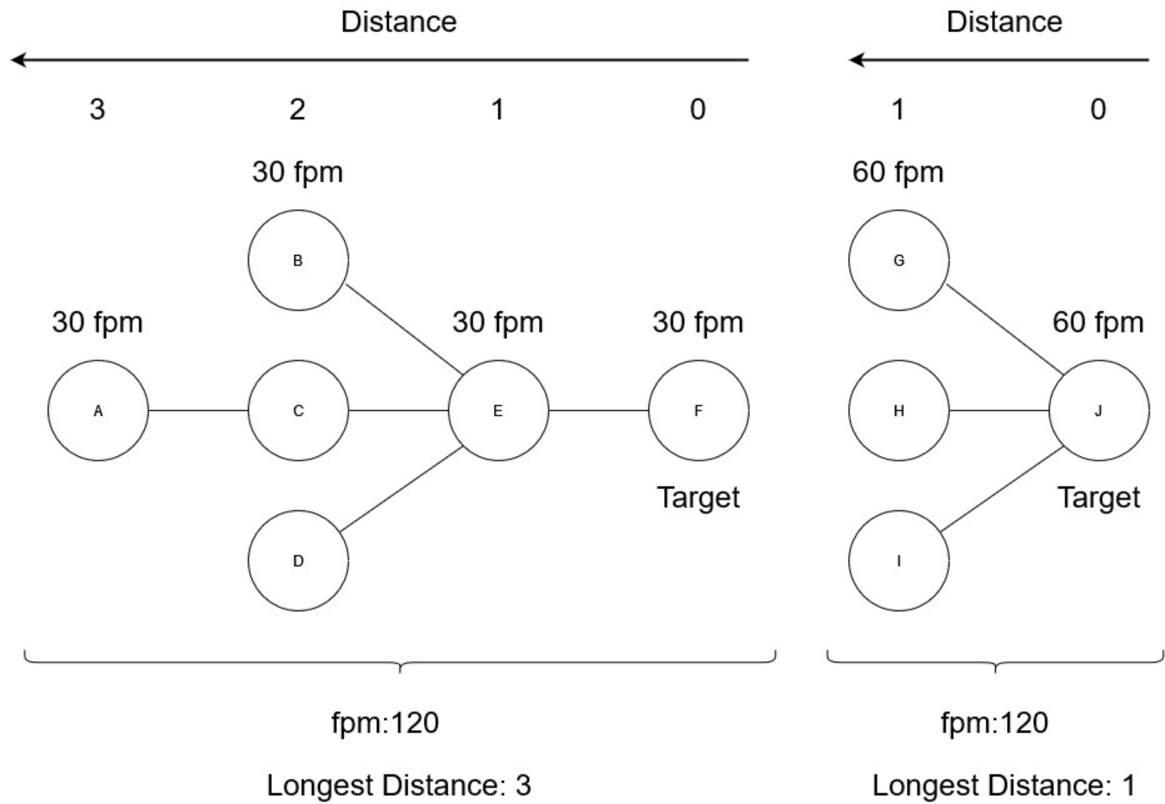


Figure 29: Due to the different graph layout, the same fpm in the payload results in distinct fpm per device. This applies for when the “offset_on” flag is set to 1.

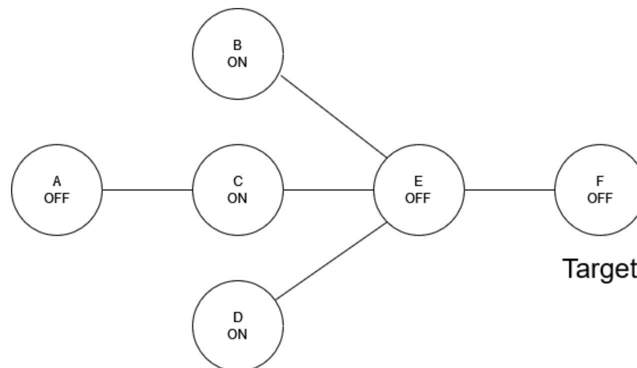


Figure 30: The devices that have the same distance to the target device is on or off at the same time. This applies for when the “offset_on” flag is set to 1.

6.3.2 Pulse

A POST to start a pulse pattern has the structure seen in figure 31 with the pattern specific variable duration. The ext_red, ext_green and ext_blue variables are, in contrast to the flash patterns string representation, made up by float values. Consequently, a 1 for any of these implies 100 percent of that color, and 0.5 implies 50 percent etc.

```
{
    "pattern_type" : "pulse",
    "target_id" : "ae08c6b4-c1e9-4c95-ab47-44e977a62a33D",
    "pattern_running" : 1,
    "ext_red" : 1,
    "ext_green" : 0,
    "ext_blue" : 0,
    "scheduled" : 1,
    "scheduled_time" : "15:30:00",
    "offset_on" : 1,
    "duration" : 1
}
```

Figure 31: The structure of the payload when scheduling a pulse pattern at 15:30 with a color intensity of 100 percent red that should pulse to the device with ID ae08c6b4-c1e9-4c95-ab47-44e977a62a33D.

As with the flash pattern the pattern specific variable, in this case duration, has two contexts depending on the "offset_on" flag. If the flag has the value of 0, all devices pulse at the same time with a step size of the max LED intensity (255) divided by the duration. Hereby, the output value y for the LED depends on how long the pulse has been running for and the resolution i.e., step size. Furthermore, the final color intensity also depends on the percentage for each color. The code snippet in figure 32 shows how the LED output is calculated.

```
while(now <= duration && pattern_running)
{
    y = step_size * now;
    int red = (int)(y * r);
    int blue = (int)(y * b);
    int green = (int)(y * g);
```

Figure 32: The calculations for the LED output. The r, g, and b variables are the percentages of the respective color derived from the payload.

Additionally, if the "offset_on" flag has a value of 1, each start time for a pulse from a device perspective has an offset depending on its distance to the target device. The calculations for the offset and the initial sleep time can be seen in the code snippets in figure 33 and figure 34, respectively. Consequently, this offset results in a pulse duration from the global graph perspective that is $2 * duration$ seconds, visualized in figure 35.

```
float offset = (float)dist_to_target / (float)longest_dist;
```

Figure 33: Offset calculation for the device.

```
if(is_offset)
{
    float offset = json_real_value(json_object_get(led_root, "offset"));
    pp->initial_sleep = (1 - offset) * pp->duration;
}
```

Figure 34: The time before the pattern starts on the device depends on the offset value and the duration of the pulse.

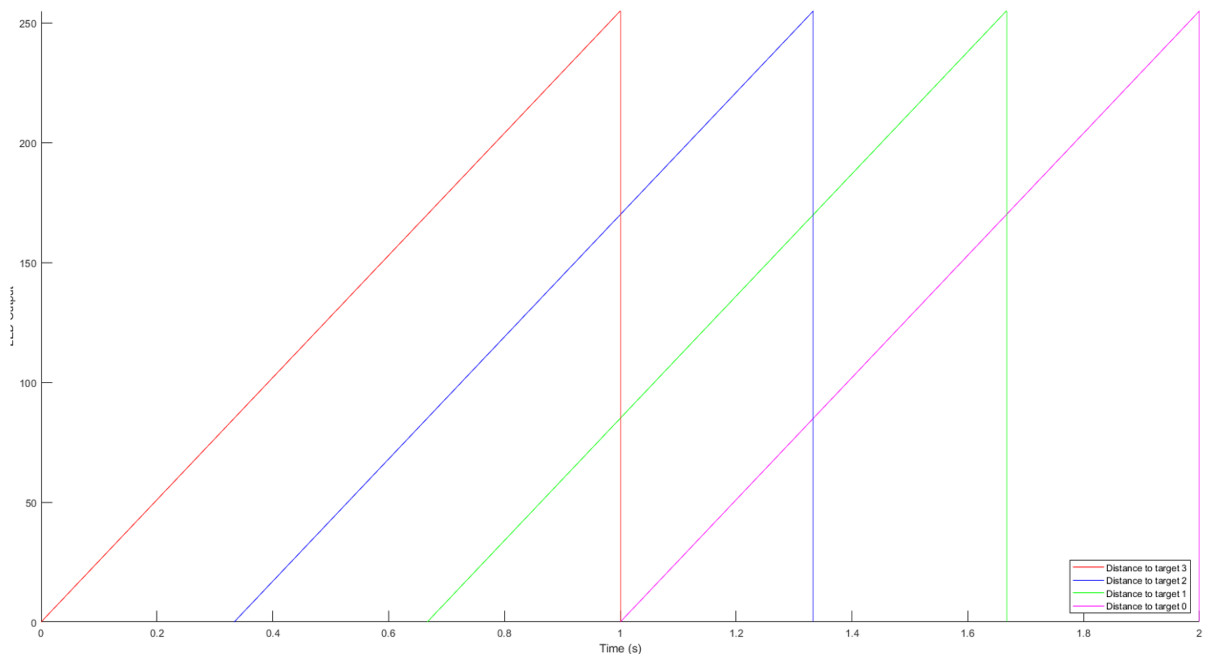


Figure 35: LED output values for four different devices with distance to target ranging from 3 to 0. The “offset_on” flag has a value of 1 and the duration is 1 second. Note that the target device starts when the pulse for the device with the longest distance has finished. Consequently, the pulse duration lasts for 2 seconds in the perspective of the layout of the devices.

6.3.3 Sine

The sine pattern has the specific variables period and phase, where the phase variable is the shift of the sinus wave expressed in radians. As with the pulse pattern, the ext_red, ext_green,

and ext_blue variables are made up of float values. The structure for a POST to start a sine pattern can be seen in figure 36.


```
{
    "pattern_type" : "sine",
    "target_id" : "ae08c6b4-c1e9-4c95-ab47-44e977a62a33D",
    "pattern_running" : 1,
    "ext_red" : 1,
    "ext_green" : 0,
    "ext_blue" : 0,
    "scheduled" : 1,
    "scheduled_time" : "15:30:00",
    "offset_on" : 1,
    "period" : 1,
    "phase" : 0
}
```


Figure 36: The structure of the payload when scheduling a sine pattern at 15:30 with a color intensity of 100 percent red that indicates a path to the device with ID ae08c6b4-c1e9-4c95-ab47-44e977a62a33D.

For an offset_on flag value of 0, all the devices in the layout have the same phase shift, thus producing the same output values at the same time, as visualized in figure 36. Opposed to a zero value, a value of 1 calculates a new phase shift considering the location of the device in respect to the target device. The code for calculating the phase shift and the output LED values for the graph, previously displayed in figure 26, can be seen in figure 37 and figure 39, respectively.

```
if(offset_on)
{
int nbr_of_devices = longest_dist + 1;
float phase_shift_per_device = ((2 * M_PI) / nbr_of_devices);
phase = phase + dist_to_target * phase_shift_per_device;
}
```

Figure 37: Code snippet displaying how the phase shift depends on the location relative to the target device.

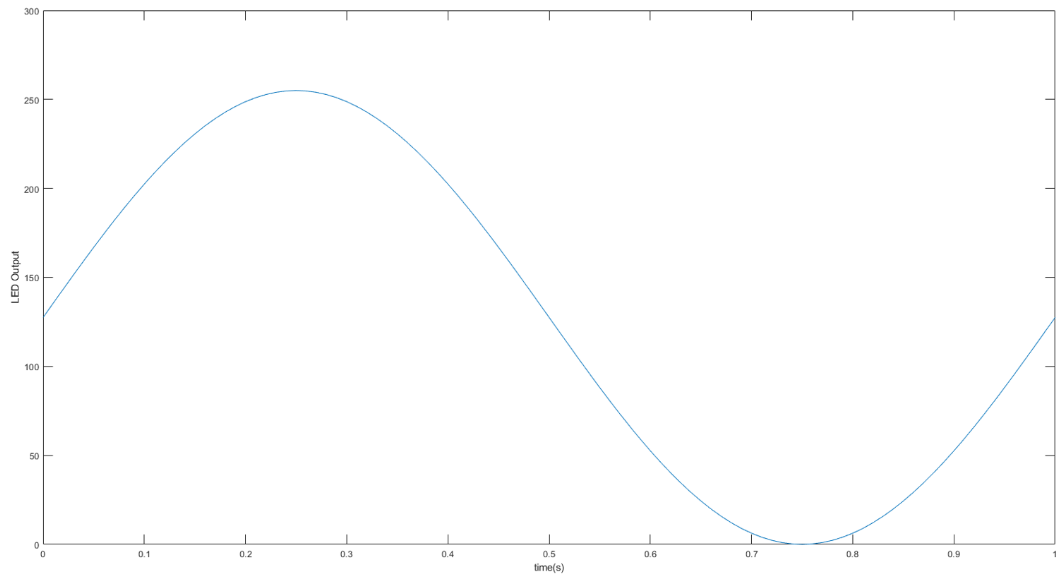


Figure 38: The output values for all devices over time with a period of one second and a phase shift of zero radians.

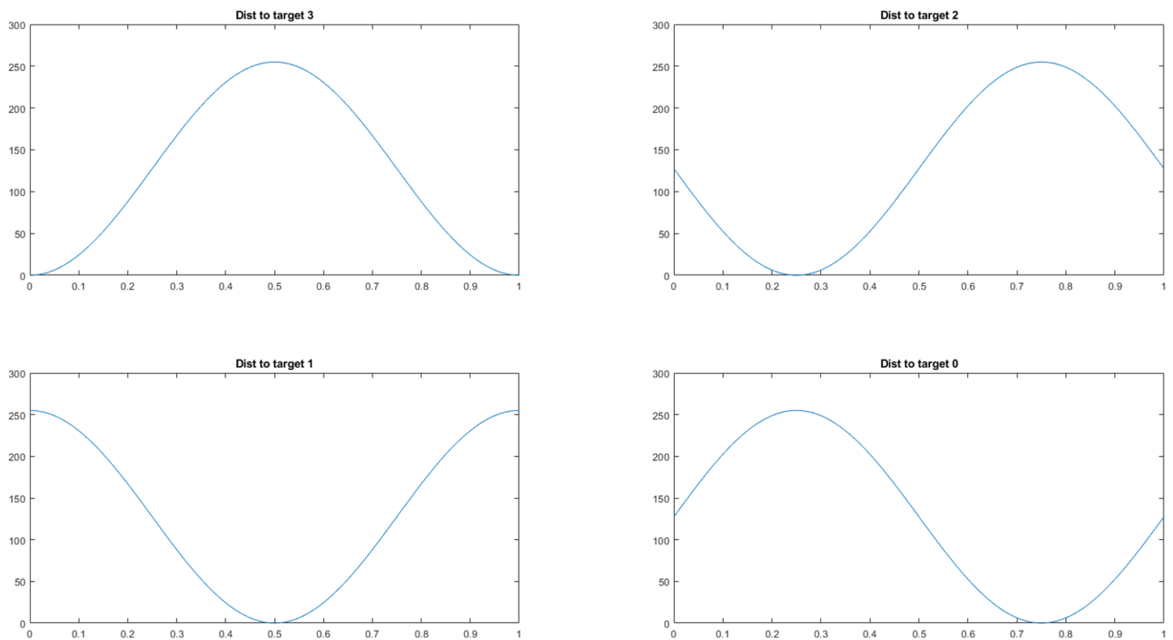


Figure 39: The output values for the devices over time with a period of one second with an offset_on flag value of 1 and the graph layout from figure 26.

6.4 API

The API was constructed by setting up an Apache HTTP server as a reverse proxy, which sent all requests to a SOAP server that had all the callback methods that were needed to control the program via a graphical user interface. The SOAP server was written with the help of the libsoup [25] library as can be seen in figure 40. It is important to note that

“proxy_path” is the path to the socket our Apache HTTP server will deliver requests. Another important point is the handlers that are added to serve different types of requests. As can be seen in figure 41, once such handler takes care of delivering the current graph layout to the user’s browser by manipulating the head and body of the SoupMessage that was sent to the SOAP service. In figure 42, we can see the two methods that are used to help the handlers appending the correct information, so the browser can later unpack this envelope and parse it without faults.

```
void *
http_server_new()
{
  printf("wf-http_server::http_server_new: %s\n", "Starting http server");

  const char *server_header = SOUP_SERVER_SERVER_HEADER;
  const char *simple_httpd = "wf-audio-light";
  const char *proxy_path = "/var/run/wf-audio-light/httpproxy";
  const char *end_point_root = "/";
  const char *end_point_layout = "/api/layout";
  const char *end_point_led = "/api/led";
  const char *end_point_index_css = "/css/index";
  const char *end_point_index_js = "/js/index";
  const char *end_point_members = "/api/members";

  SoupServer *server = soup_server_new(server_header, simple_httpd, NULL);
  soup_server_add_handler (server, end_point_root, handle_request_root, NULL, NULL);
  soup_server_add_handler (server, end_point_layout, handle_request_layout, NULL, NULL);
  soup_server_add_handler (server, end_point_led, handle_request_led, NULL, NULL);
  soup_server_add_handler (server, end_point_index_css, handle_request_index_css, NULL, NULL);
  soup_server_add_handler (server, end_point_index_js, handle_request_index_js, NULL, NULL);
  soup_server_add_handler (server, end_point_members, handle_request_members, NULL, NULL);
  /*
  * https://libsoup.org/libsoup-3.0/SoupServer.html#SoupServerListenOptions
  */
  SoupServerListenOptions options = 0;
  GError *error = NULL;
  GSocket *socket = get_uds_socket(proxy_path, NULL, "www");
  g_assert_nonnull(socket);

  soup_server_listen_socket(server, socket, options, &error);
  //soup_server_listen_all(server, 80, options, &error);
  g_object_unref(socket);
  g_assert_no_error(error);
```

Figure 40: Code snippet displaying how the SOAP server is created.

In figure 41, the handler that takes care of requests for the graph layout, checks the SoapMessage to see if it is a get or post request. When it is a get request, the handler simply replies with the layout graph file that currently exists on the local drive. If it however is a post request, the handler will first publish the new layout graph to all other devices that are

subscribed to the related topic, via the MQTT broker and after that it will reply with the local layout graph file.

```
static void
handle_request_layout(SoupServer       *server,
                      SoupMessage       *msg,
                      const char        *path,
                      GHashTable        *query,
                      SoupClientContext *client,
                      gpointer           user_data)
{
  printf("%s\n", "wf-http_server::handle_request_layout: Called");
  if(strcmp(path, "/api/layout") == 0)
  {
    printf("wf-http_server::handle_request_layout::path=wayfinder/api/layout::message->method=%s\n",  msg->method);
    if(strcmp(msg->method, "GET") == 0)
      reply(msg, get_json_root("/etc/dynamic/audio-site/wf-layout-graph.json"));
    else if(strcmp(msg->method, "POST") == 0)
    {
      mqtt_publish(ss.wfclient, 0, ss.wfclient->mqtt_topics.TOPIC_LAYOUT_GRAPH, msg->request_body->length, msg->request_body->data, 2, false);
      reply(msg, get_json_root("/etc/dynamic/audio-site/wf-layout-graph.json"));
    }
  }
  else
  {
    printf("%s\n", "wf-http_server::handle_request_layout: unknown path");
    json_t *error_json = json_object();
    json_object_set_new(error_json, "error", json_string("unknown path"));
    reply(msg, error_json);
  }
}
```

Figure 41: Code snippet of a request handler that helps the user interact with the graph layout.

It is in figure 42, we can see that by appending what type of content is included in this message, the browser will understand how it should parse the message.

```
void
json_to_body(SoupMessageBody *body, json_t *jdata)
{
  if (jdata != NULL) {
    /* Better json_dumpb exists in jansson 2.11. */
    gchar *buf = json_dumps(jdata, JSON_INDENT(2) | JSON_ENCODE_ANY);
    size_t size = strlen(buf);
    soup_message_body_append(body, SOUP_MEMORY_TAKE, /* SOUP_MEMORY_TAKE,
                                                        SOUP_MEMORY_STATIC */
                             buf, size);
    //printf("wf-http_server::handle_request_root::json_to_body::buf=%s\n", buf);
    //printf("wf-http_server::handle_request_root::json_to_body::message->response_body->data=%s\n", body->data);
  }
}

static void
reply(SoupMessage *msg, json_t *response)
{
  json_to_body(msg->response_body, response);
  soup_message_headers_append(msg->response_headers, "Content-Type", "application/json");
  soup_message_headers_append(msg->response_headers, "Cache-Control", "no-store");
  /* soup_message_headers_append(msg->response_headers, "Access-Control-Allow-Origin", "*");
  soup_message_headers_append(msg->response_headers, "Access-Control-Allow-Methods", "POST, PUT, GET, OPTIONS"); */
  soup_message_set_status(msg, SOUP_STATUS_OK);
  json_decref(response);
}
```

Figure 42: Code snippet of methods that are used to help the handlers parsing information.

Additional request handlers were added to serve requests for HTML, CSS and JavaScript files that were needed to give the user control of the program via a graphical user interface.

6.5 GUI

Figure 43 shows the user interface that is used to control the program and is accessible via a browser. The left table in figure 43 contains information that is related to the LED configuration file. When a user loads the page, the user will get the latest configuration filled into the fields. By changing values in the different fields and pressing the submit button, the program will get a new LED configuration file and a new pattern will start. In figure 43 the middle division contains a graphical representation of the current graph layout. Users can interact with the graph layout by adding or removing devices by choosing devices in the rightmost tables.

Figure 43: A screenshot of the graphical user interface that is used to control the program.

6.6 Overview How it works

The first step of initiating the program consists of creating a struct that can be seen in appendix 4, this struct keeps track of relevant information that is needed to create synchronized patterns across the devices. Subsequently, is to make sure the client can connect to the broker as can be seen in appendix 3, in this step the client will also subscribe to the topics seen previously in figure 22. The last initiation step is to create a thread that keeps track of what the led pattern should be currently doing.

After the initiation phase the MQTT client will wait until it receives a message on a topic channel it is subscribed to. When the client receives a message, it will parse the message for a topic and proceed with a course of action that is decided by the topic, as can be seen in appendix 5. If the topic is “wf-audio-light/target”, the client will parse the message for its

payload and send it to the pattern module, which can be seen in the sequence diagram of figure 44. On receiving the payload, the pattern module will call the utility module to write the message onto a local file and after that it will trigger the changes by changing the state of the pattern algorithm.

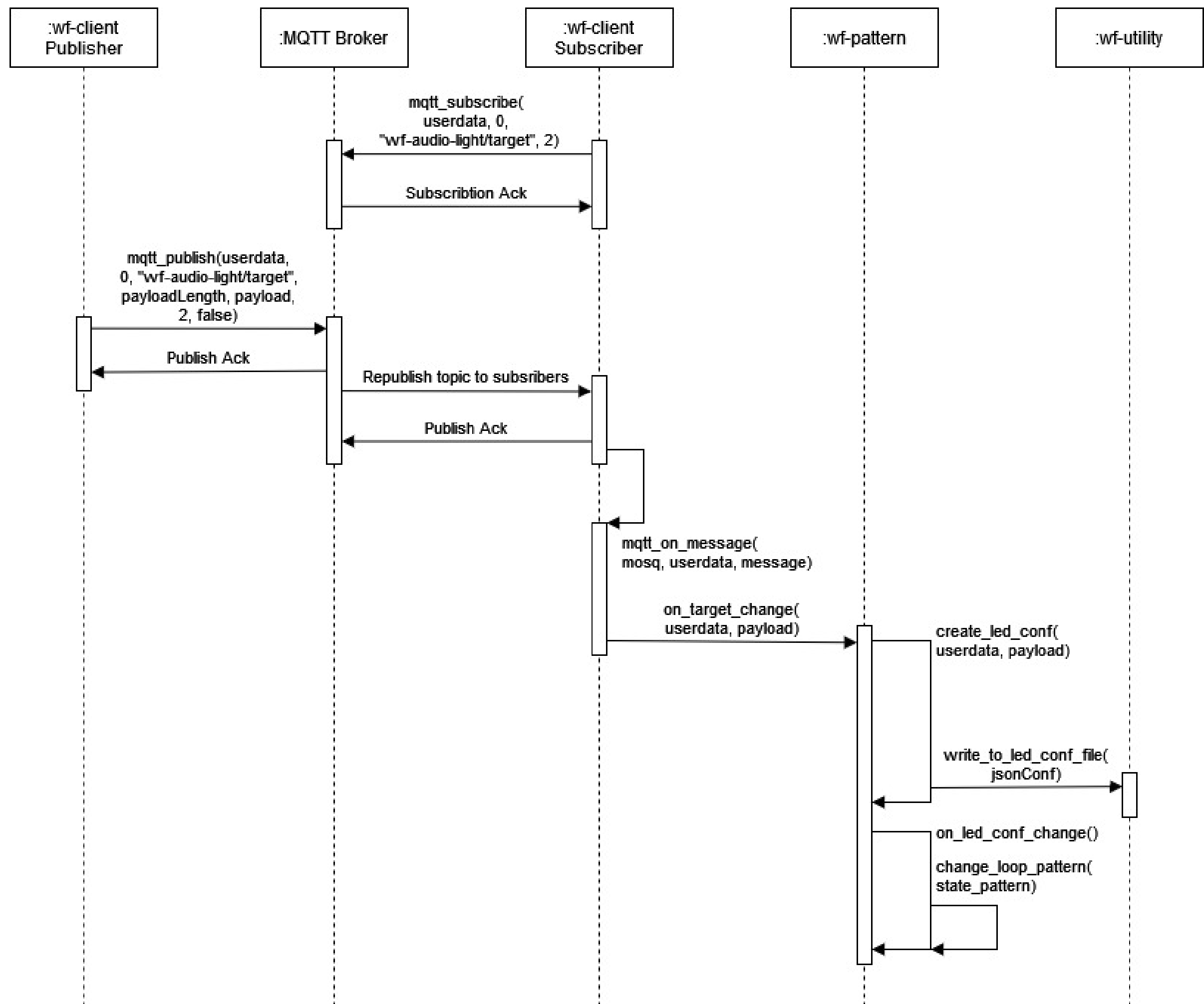


Figure 44: This figure shows the sequence that takes place when a change of target in a pattern is published.

If the client instead parses a message with the topic of “wf-audio-light/layout/edges”, it will use the utility module to write the message payload onto a local file and then tell the pattern module that a change in the layout has occurred as can be seen in the sequence diagram of figure 45. Once the pattern module is aware of this change, it will use the graph module to recalculate the distances based on the graph. This is important because the pattern algorithms

use these distances to calculate when each device should activate its LED and when it should turn it off.

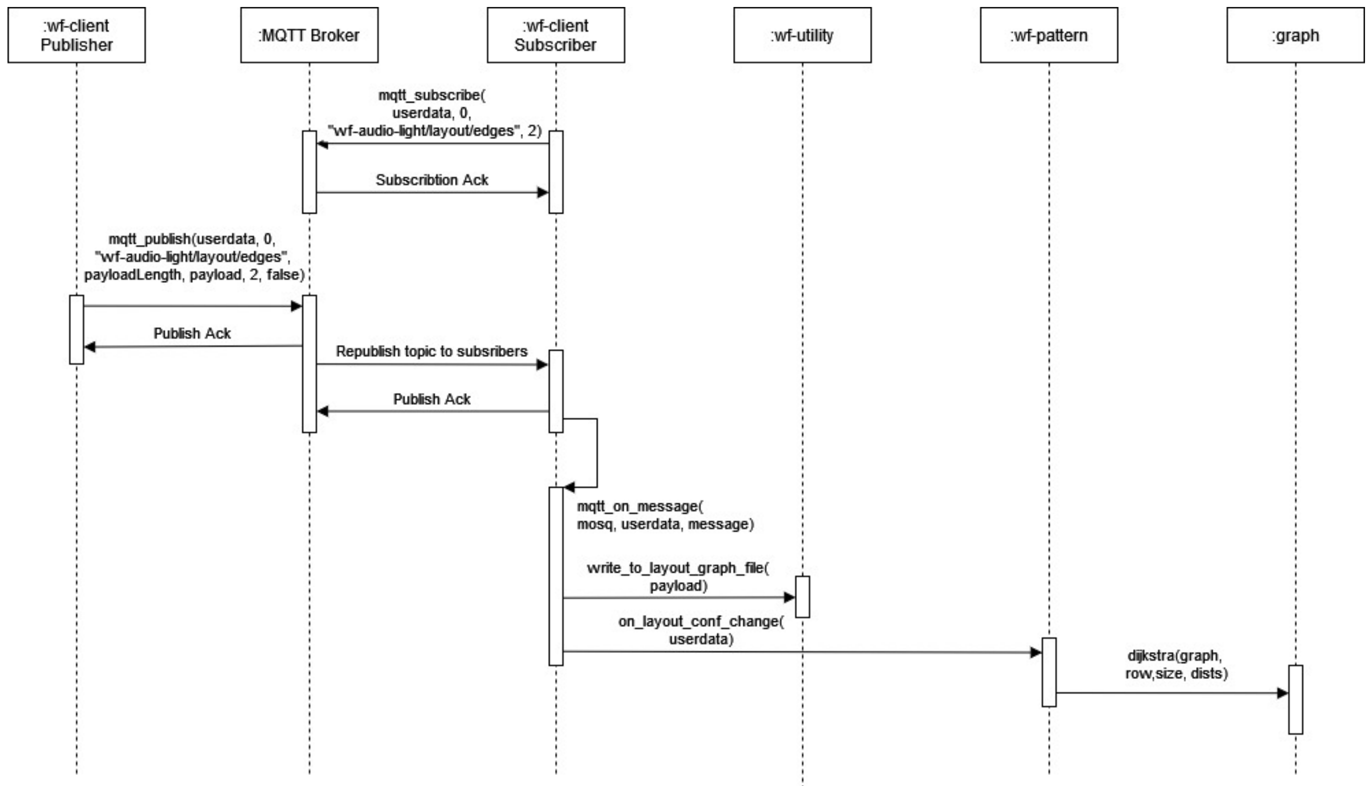


Figure 45: This figure shows the sequence that takes place when a layout change is published.

The second step of initiation the program consists of starting the SOUP API that will handle the requests that are sent via the reverse proxy Apache HTTP server.

7 Conclusion

The aim of this study was to explore and identify a method for synchronizing LEDs and audio with multiple C1410 minion speakers on an IP network. Specifically, it was of interest to

expand the area of use for the built-in LEDs and PIR sensor. Subsequently, the study intended to answer the following research questions.

1. What synchronization method in respect to cost, security and precision is suitable for LED synchronization on the C1410 speaker?
2. Will synchronization of LEDs on the speakers expand the possible area of use in respect to the built-in components (PIR sensors and LEDs)?

The study established three main objectives. Firstly, two synchronization protocols were researched, NTP and PTP. Secondly, after considering the cost and effort for implementing the different protocols, it was decided to conduct experiments with NTP. The goal of the experiments was to find and validate essential precision criteria for synchronization such as offset and clock skew. Thirdly, a suitable IoT messaging protocol was chosen based on the company's protocol in use as well as other criteria such as low delivery delays.

While PTP theoretically surpasses NTP concerning the precision aspect, the results from the experiment confirmed that NTP was a suitable synchronization protocol for the prototype. In contrast, NTP is more cost efficient than PTP because the NTP service is already installed and configured on the speakers. Thus, NTP does not need any further development as it would be the case for PTP. Finally, NTP was deemed secured with regards to the introduction of NTS and that research and development in the security area of NTP is still on-going.

MQTT worked well as an IoT messaging protocol for the protocol and no abnormalities in delay were noticed during the development phase. However, it should be noted that the development was conducted on a limited number of speakers. Furthermore, the use of MQTT would most likely ease integration to existing code and thus be more cost-efficient than other available IoT protocols.

In relation to the second research question, the prototype indicates that displaying paths to an arbitrary device is possible if the LEDs are synchronized, thus expanding the area of use for the LEDs. However, it should be acknowledged that during the development phase the devices were always in close proximity of each other. Due to time constraints, no suitable use for the PIR sensor was found for the prototype.

7.1 Ethical Aspects

According to David B. Resnik [28], openness is an important aspect when conducting research, that is, society should benefit from the results. This is further emphasized by the engineering code of honor, where it is stated that “The engineer should in its line of work feel a personal responsibility that the technique is used in such a way that it benefits humans, environment and society" [4]. However, Resnik [28] also states that the aspect of openness in some cases can be neglected to protect intellectual property. Therefore, when presenting the findings of the study, it’s important to be transparent with the company about what results will be presented publicly. Concretely, letting the company whose intellectual property may be at risk for exposure have a chance to read through the study and raise their opinion for possible leaks before it’s published. Further support for this can be found once again in the engineering code of honor, “The engineer should show full loyalty to the employer and colleagues. Difficulties in this aspect should be brought up in an open discussion, first and foremost at the workplace” [5]. Hence this report has been given to the supervisor at Axis to receive approval before published.

8 Future Development & Research

The layout graph for the devices is currently sent in manually to the API endpoint either through the terminal with the curl command or through the graphical interface. This could in the future be automated by, for instance, sound localization by utilizing the speakers’ two-way microphone.

[4] Ingenjören bör i sin yrkesutövning känna ett personligt ansvar för att tekniken används på ett sätt som gagnar människa, miljö och samhälle.

[5] Ingenjören bör visa full lojalitet mot arbetsgivare och arbetskamrater. Svårigheter härvidlag bör tas upp till öppen diskussion, i första hand på arbetsplatsen.

Due to time-constrains no suitable use cases were found for the PIR sensor. More precisely, no solution to handle target changes when two or more sensors were active at the same time was found. Therefore, possible solutions for the built-in PIR sensor could be researched in the aspect of how to decide on the target device when two or more PIR sensors detects movement.

Furthermore, the study acknowledges that the access to a limited number of speakers during the development phase have raised the question how well a path indication would work in a real-life scenario. For example, in the set up used during development, the devices were always in close proximity of each other. Therefore, future research could explore the maximal distance of the devices in a more realistic environment in the aspect of path indication. Additionally, the study acknowledges that the limited number of devices also raise concern regarding how well the synchronization would perform with more devices connected to the broker. For example, there might be a noticeable delay between devices in receiving target and pattern data, hence stress testing the MQTT messaging could be conducted.

9 Terminology

Apache

Apache is open-source software that is used to setup webservers.

API

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.

Bitbake

Bitbake is a task execution system that helps to automate the process of assembling entire embedded Linux distributions.

Clock Drift

Clock drift is changes in frequency of the crystal oscillation over time.

Clock Offset

Clock offset is the time difference between two clocks.

Clock Skew

Clock skew is the frequency difference between two clocks.

Image

Images are the binary output that runs on specific hardware.

JSON

JSON is the acronym of JavaScript Object Notation, which is a text format that represents objects or data structures.

MQTT

MQTT is a publish and subscribe protocol for messaging between devices that runs on top of TCP.

NTP

NTP is the acronym of The Network Time Protocol, which is a protocol that is used to synchronize clocks over the internet.

PoE

PoE is the acronym of Power over Ethernet, which is the technology that is used to pass electric power over ethernet cables to powered devices.

PTP

PTP is the acronym of Precision Time Protocol, which is a protocol that is used to synchronize clocks throughout a computer network.

SOAP

SOAP is a messaging protocol that is used for exchanging structured information when creating web services in computer networks.

10 References

[1] Axis Communications AB. [Internet] About axis—axis.com. [cited 19 December 2021] Available from : https://www.axis.com/about-axis

[2] Axis Communications AB. [Internet] Annual review & sustainability report. [cited 19 December 2021] Available from: https://www.axis.com/files/brochure/axis_annual_review_and_sustainability_report_2020.pdf.pdf

[3] Francisco Tirado-Andrés, Alba Rozas, and Alvaro Araujo. A methodology for choosing time synchronization strategies for wireless IoT networks. Sensors, 19(16), 2019.

[4] Workday. [Internet] Axis.wd3.myworkdayjobs.com. [cited 19 December 2021] Available from: https://axis.wd3.myworkdayjobs.com/en-US/External_Career_Site/job/Sweden---Lund/Master-Thesis---Wayfinder-using-audio-and-light_R-117601

[5] Chowdhury D. "NextGen Network Synchronization". 1st ed. Springer, Cham; 2021. DOI: https://doi.org/10.1007/978-3-030-71179-5

[6] M. Coustans *et al.*, "A 32kHz Crystal Oscillator Leveraging Voltage Scaling in an Ultra-Low Power 40NA Real-Time Clock," *2018 31st IEEE International System-on-Chip Conference (SOCC)*, 2018, pp. 308-313, DOI 10.1109/SOCC.2018.8618534.

[7] [RFC2330] Paxson. V., Almes., G., Mahdavi. J., Mathis. M., "Framework for IP performance Metrics", RFC 2330, DOI 10.17487/RFC2330, May 1998

[8] [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010

[9] How does it work? [Internet] Ntp.org [cited 15 February 2022] Available from: http://www.ntp.org/ntpfaq/NTP-s-algo.htm#Q-ALGO-BASIC-SYNC

[10] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," in IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008), vol., no., pp.1-499, 16 June 2020, DOI 10.1109/IEEESTD.2020.9120376.

[11] MQTT Version 5.0 [Internet]. Docs.oasis-open.org. 2019 [cited 30 March 2022]. Available from: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

[12] MQTT Client and Broker and MQTT Server and Connection Establishment Explained - MQTT Essentials: Part 3 [Internet]. MQTT Essentials - a ten-part blog series. 2019 [cited 30 March 2022]. Available from: https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/

[13] MQTT Publish, Subscribe & Unsubscribe - MQTT Essentials: Part 4 [Internet]. MQTT Essentials - a ten-part blog series. 2019 [cited 30 March 2022]. Available from: https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe/

[14] The Meson Build system [Internet]. Mesonbuild.com. 2022 [cited 19 April 2022]. Available from: https://mesonbuild.com

[15] The Yocto Project Development Environment [Internet]. “Introducing the Yocto Project”**s** [cited 19 April 2022]. Available from: https://docs.yoctoproject.org/overview-manual/yp-intro.html

[16] Yocto Project Concepts [Internet]. [cited 19 April 2022]. Available from: https://docs.yoctoproject.org/overview-manual/concepts.html#openembedded-build-system-concepts

[17] BitBake Concepts [Internet]. [cited 19 April 2022]. Available from: https://docs.yoctoproject.org/bitbake/bitbake-user-manual/bitbake-user-manual-intro.html#concepts

[18] [RFC8259] Bray, T., Ed., “The JavaScript Object Notation (JSON) Data Interchange Format”, RFC 8259, DOI 10.17487/RFC9259, December 2017

[19] JSON [Internet]. Json.org. 2022 [cited 9 May 2022]. Available from: https://www.json.org/json-en.html

[20]. Kovácsházy T, Ferencz B. Performance Evaluation of PTPd, a IEEE 1588 implementation, on the x86 Linux platform for Typical Application Scenarios [Internet]. Budapest: Budapest University of Technology and Economics; 2012. Available from: https://ieeexplore-ieee-org.ludwig.lub.lu.se/stamp/stamp.jsp?tp=&arnumber=6229387&tag=1

[21]. Wytrębowicz J, Cabaj K, Krawiec J. Messaging Protocols for IoT Systems—A Pragmatic Comparison. 2021, DOI 10.3390/s21206904.

[22] Reverse Proxy Guide - Apache HTTP Server Version 2.4 [Internet]. Httpd.apache.org. 2022 [cited 14 May 2022]. Available from: https://httpd.apache.org/docs/2.4/howto/reverse_proxy.html

[23] Apache HTTP server [Internet]. [cited 14 May 2022]. Available from: https://httpd.apache.org/docs/2.4/

[24] [RFC4227] O'Tuathail, E., Clipcode.com., Rose, M., Dover Beach Consulting, Inc, "Using the Simple Object Access Protocol (SOAP)", RFC 4227, DOI 10.17487/RFC4227, Januari 2006

[25] Libsoup -3.0 [Internet]. [cited 14 May 2022]. Available from: https://www.libsoup.org/libsoup-3.0/index.html

[26] [RFC5906] Haberman, B., Ed., and D. Mills, "Network Time Protocol Version 4: Autokey Specification", RFC 5906, DOI 10.17487/RFC5906, June 2010

[27] [RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020

[28] Resnik D. Openness versus Secrecy in Scientific Research [Internet]. 2006. [cited 26 May 2022] Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2991133/pdf/nihms33449.pdf

11 Appendix

[1]

```
char* sample_data()
{
    char command[] = "ntpctl -s all";
    char poll_intervall[255];
    char next_poll[255];
    char offset[255];
    char delay[255];
    char jitter[255];

    FILE *f = popen(command, "r");
    if(f == NULL)
    {
        printf("Failed to create stream\n");
        pclose(f);
        exit(-1);
    }

    extract_data(f, offset, delay, jitter, poll_intervall, next_poll);

    printf("Next Poll: %s\n", next_poll);
    printf("Poll Intervall: %s\n", poll_intervall);
    printf("Delay: %s\n", delay);
    printf("Offset: %s\n", offset);
    printf("Jitter: %s\n", jitter);
    pclose(f);

    write_to_file(poll_intervall, offset, delay, jitter);
    char *min_sleep = strtok(next_poll, "s");
    return min_sleep;
}
```

[2]

```
void extract_data(FILE *f, char *offset, char *delay, char *jitter, char *poll_intervall, char *next_poll)
{
   /** Example output of the 'ntpctl -s all' command
    * |
    * |1/1 peers valid, clock synced, stratum 5
    * |
    * |peer
    * |  wt tl st  next  poll          offset       delay      jitter
    * |172.25.8.44
    * | 1 10  4   13s   31s        -0.374ms     1.149ms     0.903ms
    */

   //Retrieve the last line
   char data[512];
   while(fgets(data, sizeof data, f) != NULL);
   printf("%s\n",data);
   //Move to token (next)
   char *token = strtok(data, " ");
   if(token == NULL)
   {
       printf("Can't retrieve data. No tokens to read.");
       pclose(f);
       exit(-1);
   }

   //Example output of 'ntpctl -s all' command when synced to peer, asterix '*' is added
   /* |* 1 10  4   13s   31s        -0.374ms     1.149ms     0.903ms*/
   if(strcmp(token, "*") == 0)
   {
       token = strtok(NULL, " ");
   }

   for(int i = 0; i < 2; ++i)
   {
      token = strtok(NULL, " ");
   }

   strcpy(next_poll, strtok(NULL, " "));
   strcpy(poll_intervall, strtok(NULL, " "));
   strcpy(offset, strtok(NULL, " "));
   strcpy(delay, strtok(NULL, " "));
   strcpy(jitter, strtok(NULL, " "));
}
```

[3]

```
/* Initial setup */
struct wfclient client = {NULL, {" ", " ", " ", " ",0, 0}, 0, 0, {"", "", ""}, NULL, NULL, NULL};

if(!init(&client))
    return -1;

set_callbacks(&client);
mqtt_connect(&client);
if(client.connect_status)
{
    fprintf(stderr, "Unable to connect.\n");
    return -1;
}
/* Create the thread that will handle the LED-pattern */
pattern_loop_start();
```

[4]

```c
struct site_settings
{
    char device_id[255];
    char host[255];
    char site_id[255];
    char mqtt_PSK[255];
    int port;
    int isActiveBroker;
};

struct topics
{
    char TOPIC_LAYOUT_GRAPH[250];
    char TOPIC_LEDCONF[250];
    char TOPIC_TARGET[250];
};

struct wfclient
{
    struct mosquitto *mosq;
    struct site_settings settings;
    int connect_status;
    int is_connected;
    struct topics mqtt_topics;
    struct Graph *graph;
    int* distances;
    size_t** distance_matrix;
};
```

[5]

```c
void mqtt_on_message(struct mosquitto *mosq, void *userdata, const struct mosquitto_message *message)
{
    struct wfclient *self = userdata;
    printf("Topic: %s, Qos: %d, Payload: %s\n", message->topic, message->qos, (char *)message->payload);
    struct mosquitto_message copy[message->payloadlen];
    mosquitto_message_copy(copy, message);

    //Handle topics
    if(!strcmp(copy->topic, self->mqtt_topics.TOPIC_LAYOUT_GRAPH))
    {
        //Write layout graph to file
        write_to_layout_graph_file(copy->payload);
        //Trigger layout change
        on_layout_conf_change(self);
    }
    else if(!strcmp(copy->topic, self->mqtt_topics.TOPIC_TARGET))
    {
        //Calculate led_conf_file
        on_target_change(self, copy->payload);
    }
}
```

[6]

```
float blink_per_second = (float)fpm/60.0f;

float offset = 0;
float sleep_time;
float awake_time;

if(offset_on)
{
    sleep_time = (float)(longest_dist)/blink_per_second;
    awake_time = (float)1/blink_per_second;
    offset = ((float)(longest_dist - dist_to_target) / (float)(longest_dist)) * (sleep_time);
}
else
{
    sleep_time = 1/(blink_per_second*2);
    awake_time = 1/(blink_per_second*2);
}
```

[7]

```
mqtt_connect(struct wfclient *self)
{
    if(self->is_connected)
        return;

    int port;
    char *address;
    char *identity;
    char *mqtt_psk;
    int result = 0;

    port = self->settings.port;
    address = self->settings.host;
    identity = self->settings.site_id;
    mqtt_psk = self->settings.mqtt_PSK;

    char psk_mosquitto[2 * PSK_MOSQUITTO_SIZE + 1];
    int sourceindex = 0;
    int destindex = 0;
    while(destindex < PSK_MOSQUITTO_SIZE * 2 + 1)
    {
        if(mqtt_psk[sourceindex] != '-')
        {
            psk_mosquitto[destindex] = mqtt_psk[sourceindex];
            destindex++;
        }
        sourceindex++;
    }

    //Setup TLS-PSK
    result = mosquitto_tls_psk_set(self->mosq, psk_mosquitto, identity, "PSK-AES128-CBC-SHA");
    if(result != MOSQ_ERR_SUCCESS)
    {
        mosquitto_destroy(self->mosq);
        fprintf(stderr,"error tls_psk_set:%s\n", "mosquitto_tls_psk_set failed");
        return 1;
    }

    self->connect_status = mosquitto_connect_async(self->mosq,address,port,60);
    if(self->connect_status != MOSQ_ERR_SUCCESS) {
        mosquitto_destroy(self->mosq);
        fprintf(stderr, "Error connect_async: %s\n", mosquitto_strerror(self->connect_status));
        return 1;
    }

    result = mosquitto_loop_start(self->mosq);
    if(result != MOSQ_ERR_SUCCESS){
        mosquitto_destroy(self->mosq);
        fprintf(stderr, "Error loop_start: %s\n", mosquitto_strerror(result));
        return 1;
    }

    return 0;
}
```