MASTER

Time synchronization in IoT lighting control

Beke, T.

Award date:
2017

Link to publication
Time synchronization in IoT lighting control

Master Thesis

Beke, Tibor

Supervisors:
  dr. T. Özgelebi
  dr. ir. P.H.F.M. Verhoeven
  ing. H. Stevens
  dr. ir. E.O. Dijk
  dr. ir. M.C.W. Geilen

Eindhoven, 20 December 2016
Abstract

The Internet of Things is the next evolution of the internet, a game-changer in collecting, analyzing and distributing data. The challenge taken up by the OpenAIS (Open Architectures for Intelligent Solid State Lighting Systems) project is to set the leading standard for inclusion of lighting for professional applications into IoT, with a focus on office lighting.

The advent of the IoT, the integration of the cyber and physical world, raises the questions of timing and coordination of data on a global level, including the chronological ordering of events, the asynchronous coordination of processes and new coordination strategies. These also appear in OpenAIS through data analytics, time synchronized lighting effects, or improved reliability.

Most Things are constrained in one or more ways: they have limited CPU, memory, little or no static memory. They may be challenged with unreliable or lossy communication, wireless technologies with a limited bandwidth and dynamic network topology. They may be placed in diverse settings such as in buildings, factories, outside or in natural ecosystems. The idea to bring a global notion of time to these devices presents a new area of research.

Many protocols exist to synchronize time between computers on a network. They vary for example in achieved accuracy, communication model, their dependence on special hardware, or clock correction type. Network Time Protocol (NTP) is the most commonly used protocol to synchronize computers over the internet, hence it is advantageous to apply it in this new context as well. NTP is able to synchronize time with an accuracy of tens of milliseconds over the internet and in the order of 100 microseconds in local networks. SNTP, a lightweight version of NTP, was implemented in this master project on Freescale FRDM-K64F boards. SNTP achieved a median accuracy of 7µs on the event-driven Mbed OS 3 and 58µs on the multi-threaded Mbed OS 5 synchronizing to a local NTP server over Ethernet. It is shown that the implementation performs badly in multihop 6LoWPAN networks, a commonly used open IoT networking protocol.

A new protocol is proposed, inspired by the 6LNTP protocol of Hong et al., for wireless mesh networks. The proposed Mesh Time Synchronization Protocol exploits the resource-rich 6LoWPAN Border Router to synchronize its time using NTP and disseminate the correct time in the mesh in form of radio broadcasts. The implementation achieves a median accuracy of 0.48ms, 1.01ms, 1.50ms, 2.06ms on Mbed OS 3 and 0.74ms, 1.60ms, 2.43ms, 3.28ms on Mbed OS 5 over 1, 2, 3 and 4 hops respectively.

Neither of these two protocols compensates between synchronization rounds for the crystal-oscillator originating drift. Hence, the rounds have to be repeated regularly. SNTP has little impact on modern Ethernet networks. When it is compared on an example 6LoWPAN mesh topology to the Mesh Time Synchronization Protocol, SNTP needs at least 2.5 times more transmissions.

Because of the OpenAIS requirements towards the wireless physical layer, Mesh Time Synchronization Protocol messages are encrypted on the link-layer by default. SNTP messages over Ethernet may be authenticated with symmetric or public-key cryptography.

The implemented SNTP for Ethernet networks, complemented with the Mesh Time Synchronization Protocol for 6LoWPAN-based wireless networks, complies with the OpenAIS requirements for time synchronization.
Preface

Sometimes taking time
is actually a shortcut.

What I Talk About
When I Talk About Running
Haruki Murakami

The research “Time synchronization in IoT lighting control” has been conducted to fulfill the graduation requirements of the Master’s degree in Embedded Systems at the Eindhoven University of Technology. My research questions have been formulated together with my supervisor, dr. T. Özcelebi. The research has been conducted in the OpenAIS team of Philips Lighting between April and December 2016 after a 2-month-long preparation phase in the OpenAIS team of NXP.

I would like to thank Tanir Özcelebi and Richard Verhoeven at the university for keeping me structured and organized during this long period of work. I would like to thank Henk Stevens and Esko Dijk at Philips Lighting for their detailed feedbacks and ideas. I would like to thank Marino Strik, Klaas de Waal and Ewout Brandsma for setting me off on the road.

This writing ends the challenge I have started a little more than two years ago in Berlin, but it does not stop the running.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>vii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xi</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 The OpenAIS project</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 OpenAIS architecture</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2 OpenAIS requirements for Network Access Layer</td>
<td>5</td>
</tr>
<tr>
<td>1.1.3 OpenAIS requirements for time synchronized operations</td>
<td>5</td>
</tr>
<tr>
<td>1.2 Problem statement and formalization</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Goals</td>
<td>7</td>
</tr>
<tr>
<td>1.4 Outline</td>
<td>8</td>
</tr>
<tr>
<td>2 Related work</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Time and time standards</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Network time synchronization protocols</td>
<td>10</td>
</tr>
<tr>
<td>2.2.1 Delay components in a message</td>
<td>10</td>
</tr>
<tr>
<td>2.2.2 Time protocol</td>
<td>10</td>
</tr>
<tr>
<td>2.2.3 Cristian’s algorithm</td>
<td>11</td>
</tr>
<tr>
<td>2.2.4 Network Time Protocol</td>
<td>11</td>
</tr>
<tr>
<td>2.2.5 Precision Time Protocol</td>
<td>11</td>
</tr>
<tr>
<td>2.2.6 6LNTP</td>
<td>11</td>
</tr>
<tr>
<td>2.2.7 Synchronization protocols for wireless sensor networks</td>
<td>12</td>
</tr>
<tr>
<td>2.2.8 Time synchronization protocol for OpenAIS</td>
<td>13</td>
</tr>
<tr>
<td>2.3 Clock drift correction techniques</td>
<td>15</td>
</tr>
<tr>
<td>2.4 Performance metrics for evaluation</td>
<td>16</td>
</tr>
<tr>
<td>3 SNTP</td>
<td>19</td>
</tr>
<tr>
<td>3.1 Network Time Protocol</td>
<td>19</td>
</tr>
<tr>
<td>3.1.1 NTP timestamp format</td>
<td>19</td>
</tr>
<tr>
<td>3.1.2 NTP message format</td>
<td>19</td>
</tr>
<tr>
<td>3.1.3 On-wire protocol</td>
<td>20</td>
</tr>
<tr>
<td>3.1.4 Hierarchical timeflow</td>
<td>21</td>
</tr>
<tr>
<td>3.1.5 Clock adjustment and discipline processes</td>
<td>22</td>
</tr>
<tr>
<td>3.2 NTP for constrained devices</td>
<td>22</td>
</tr>
<tr>
<td>3.2.1 SNTP</td>
<td>22</td>
</tr>
<tr>
<td>3.2.2 Software architecture</td>
<td>22</td>
</tr>
<tr>
<td>3.2.3 Implementation on Mbed OS</td>
<td>24</td>
</tr>
</tbody>
</table>

Time synchronization in IoT lighting control vii
## 4 Mesh Time Synchronization Protocol

### 4.1 Introduction

- **4.1.1 IEEE 802.15.4**
- **4.1.2 6LoWPAN**
- **4.1.3 Thread**
- **4.1.4 OpenAIS implementation**
- **4.1.5 Medium access issues in the mesh**

### 4.2 Mesh Time Synchronization Protocol: the design

- **4.2.1 Message format**
- **4.2.2 Time dissemination in the mesh**
- **4.2.3 Time request message**
- **4.2.4 Synchronization message**
- **4.2.5 Correction message**
- **4.2.6 Mesh Time Server**

### 4.3 Implementation

## 5 Evaluation

### 5.1 Measurement design and setup

- **5.1.1 FRDM-K64F board**
- **5.1.2 802.15.4 transceiver**
- **5.1.3 Setup for clock drift measurement (M1)**
- **5.1.4 Setup for SNTP accuracy over Ethernet measurement (M2)**
- **5.1.5 Setup for SNTP accuracy over 6LoWPAN measurement (M3)**
- **5.1.6 Setup for Mesh Time Synchronization Protocol accuracy over 6LoWPAN measurement (M4)**
- **5.1.7 Setup for accuracy measurement in congested network (M5)**
- **5.1.8 Setup for operating system delays measurement (M6)**

### 5.2 Results

- **5.2.1 Clock drift**
- **5.2.2 SNTP**
- **5.2.3 Mesh Time Synchronization protocol**

### 5.3 Scalability

### 5.4 Security considerations

### 5.5 Requirements towards the OpenAIS architecture

## 6 Conclusions

### 6.1 Contribution

### 6.2 Future work

## References

## Appendix

### A Software versions
List of Figures

1.1 OpenAIS network stack ................................................. 2
1.2 The basic control paradigm of the OpenAIS architecture .......... 3
1.3 Relations between functions in the application layer of the OpenAIS architecture ................................................. 3
1.4 Relations between functions in the infrastructure layer of the OpenAIS architecture ................................................. 4

2.1 Delay components between constructing the message on the source and processing it on the destination ................................................. 10
2.2 ABRACON ABS07-32.768KHZ-T crystal frequency-temperature characteristics ................................................. 15
2.3 Absolute crystal drift including random errors at various temperature conditions ................................................. 16

3.1 NTP Timestamp format ................................................. 19
3.2 The NTP algorithm ................................................. 21
3.3 The NTP architecture ................................................. 21
3.4 UML class diagram for the SNTP implementation ................. 24

4.1 IEEE 802.15.4 star and peer-to-peer network topologies .......... 26
4.2 Overview of the Thread stack ................................................. 27
4.3 OpenAIS LPR AP protocol stack ................................................. 28
4.4 OpenAIS LPR AP architecture ................................................. 29
4.5 Proposed message format for the Mesh Time Synchronization Protocol ................................................. 30
4.6 Algorithm for synchronizing time over the mesh ................. 31
4.7 Time dissemination in a multihop mesh network ................. 32
4.8 Processing of incoming SYNC messages ......................... 33
4.9 Processing of incoming CORR messages ......................... 34
4.10 Sequence diagram for Mesh Time Server restart and node restart ................................................. 35
4.11 UML class diagram for the Mesh Time Synchronization Protocol implementation ................................................. 36

5.1 Clock drift measurement setup (M1) ................................................. 38
5.2 SNTP synchronization accuracy measurement setup for Ethernet connection (M2) ................................................. 39
5.3 SNTP synchronization accuracy measurement setup for 6LoWPAN multi-hop connection (M3) ................................................. 40
5.4 Mesh Time Synchronization Protocol accuracy measurement setup for 6LoWPAN multi-hop connection (M4) ................................................. 41
5.10 SNTP synchronization accuracy with delayed server replies (asynchronous round-trip delay) (M5) ................................................. 46
5.11 SNTP synchronization accuracy with delayed server replies (M5) ................................................. 46
List of Tables

1.1 OpenAIS requirements for time synchronization ........................................ 6
2.1 Comparing network time synchronization protocols ................................. 14
4.1 802.15.4 PHY layer parameters for the standard’s supported bands .......... 26
Chapter 1

Introduction

According to Cisco [1] there will be 50 billion devices connected to the internet by 2020. The Internet of Things (IoT) is the next evolution of the internet, a game-changer in collecting, analyzing and distributing data. The importance of IoT and the fast penetration of solid-state based lighting naturally accompanies the idea of connecting luminaires to it.

The OpenAIS (Open Architectures for Intelligent Solid State Lighting Systems) project [2] is dedicated to develop standards to connect the lighting systems to the internet. The project consortium involves parties from leading lighting companies (Zumtobel, Philips, Tridonic), IoT technology leaders (ARM, NXP, Imtech) and academia (Eindhoven University of Technology, TNO-ESI). The main objectives of the project are to define and standardize an open IP-based system architecture for office lighting; to prepare the industry to make use of this architecture and offer new services and applications in office lighting; and to contribute to energy reduction by combining the IoT and LED technologies with the Smart Grid domains.

The drive behind OpenAIS is to ensure interoperability between the components from different vendors, using the concept of the Internet of Things as its foundation, hence bringing the internet protocol to every end-device. This also brings some already solved problems into a new, interesting context. Having a correct, synchronized time on our computers became obvious and generally required, even when these computers are mobile. The advent of the IoT, the integration of the cyber and physical world, raises the questions of timing and coordination of data on a global level, including the chronological ordering of events, the asynchronous coordination of processes and new coordination strategies. These appear in the lighting application through data analytics, time synchronized lighting effects, or improved reliability. Most Things are constrained in one or more ways: they have limited CPU, memory, little or no static memory, or they are operating from battery (often for years). They may be challenged with unreliable or lossy communication, wireless technologies with a limited bandwidth and dynamic network topology. They may be placed in diverse settings such as in buildings, factories, outside or in natural ecosystems. The idea to bring a global notion of time to these devices presents a new area of research.

The challenge taken up by OpenAIS also involves the notion of timing through the interconnection of areas such as energy management, maintenance or the lighting applications. This research investigates appropriate solutions in this new context. It was conducted at the Eindhoven University of Technology and it contributes to the OpenAIS project to help make better decisions in the design and implementation of the system concerning time synchronization.

1.1 The OpenAIS project

1.1.1 OpenAIS architecture

The reference architecture defined by the OpenAIS project [3] allows the creation of vendor specific, but compatible lighting control systems. OpenAIS proposes a complete network stack with support for both wireless and wired communication. The communication is IPv6 based using UDP as the
CHAPTER 1. INTRODUCTION

transportation layer, allowing any physical medium that supports these and complies with the minimum requirements listed in [1.1.2]. UDP is selected as it is a preferred communication protocol in the domain of constrained embedded devices, where memory (RAM or flash) and CPU resources are limited and the network communication is typically low-bandwidth.

On top of this, communication messages between functions and modules are carried using the CoAP protocol. CoAP is defined in [2] by the Internet Engineering Task Force (IETF) as a replacement of the HTTP for constrained Internet of Things embedded devices. CoAP shares the same REST-based semantics as HTTP, but it is coded with a higher efficiency.

On the application layer the Lightweight M2M (LWM2M) standard is used as defined by the Open Mobile Alliance (OMA). LWM2M is a framework for device management, that also allows the ability to transfer service and/or application data. LWM2M implements the interface between an LWM2M Client and an LWM2M Server using CoAP as its transport layer, with the support of various data format standards in its payload, such as JSON, TLV or plain text. The OpenAIS project proposes some additions and modifications to the OMA LWM2M framework, that also require the usage of some CoAP additions. These include the features CoAP Blockwise Transfer, the CoAP Observe (as defined in RFC7641) and the CoAP Multicast (RFC7390). The additions and modifications to the OMA LWM2M framework are to satisfy the lighting and building IoT requirements. These include the peer-to-peer unicast and group communication, so that messages do not have to travel through the server or back-end infrastructure. They also include the support for device bootstrap without relying on a pre-configuration, the local discovery of services and devices, the support for the compact data format CBOR, out-of-the-box operation functions, support for event notification and event distribution based both on pre-configured bindings and subscriptions, and extending the URI paths to 5 levels.

The communication is encrypted on the transport layer using DTLS and on the application layer using COSE. The framework allows the incorporation of IPv6 legacy systems with gateways. The networking view of the architecture is depicted in Figure 1.1.

The main control paradigm of the architecture – depicted in Figure 1.2 – can be divided into sensing, control function, and actuating. Sensing functions may communicate with several control functions, and control functions may talk to multiple actuators. A Sensor function can be a physical device such as a presence detector or light sensor, while an Actuator function represents the actual generation of light such as a luminaire. Control functions can be either integrated into a sensor or actuator hardware, or web based and implement the algorithms for the automatic behaviour. This paradigm of the core of any lighting control system.

The whole system can be decomposed of an application layer and infrastructure layer of func-
There are further functions defined in the application layer next to the previously described Sensing, Control and Actuation functions:

- DataCollect implements the collection and processing of data from sensors and other functions, and forwards them to a more permanent storage.
- The Group function provides support for the concept where multiple actuators and sensors may distribute their data to multiple control functions. An example can be where a physical switch can control a group of lights together.
- The Scene function provides support for the concept where a set of actuator settings form a scenario together. An example can be when a room can have a specific light setting for meetings.
- Gateway function, that allows the support of non-OpenAIS lighting automation systems.

The relation between these functions is depicted in Figure 1.3. These application layer functions express the domain specific functionality and will be the main area of differentiation between the

![Figure 1.2](image1.png)  
**Figure 1.2:** The basic control paradigm of the OpenAIS architecture. Taken from [3]

![Figure 1.3](image2.png)  
**Figure 1.3:** Relations between functions in the application layer of the OpenAIS architecture. Taken from [3]
different vendors implementing the OpenAIS system. In contrast to this, the infrastructure layer functions will be implemented using commodity technology. The identified functions on this layer are the following:

- Discovery function, that detects the available application layer functions in the system.
- Communication function supports a communication infrastructure between the functions of the whole system.
- Update function to allow software updates on the system.
- Security function, that provides support for authorization, authentication and secure communication across the system.
- Configuration function to configure static system parameters.
- Device function that is a container for physical device properties such as IP-addresses, power states etc.

The relation between these function is depicted in Figure 1.4.

![Figure 1.4: Relations between functions in the infrastructure layer of the OpenAIS architecture. Taken from 3.](image-url)
1.1.2 OpenAIS requirements for Network Access Layer

The main driving idea behind the OpenAIS architecture is to develop the Internet of Lights. This incorporates ideas such as to use existing IT and/or IoT frameworks by using IP protocol up to the final nodes and get rid of translating gateways. The goal is to make lighting part of the building management framework, open up control for the cloud, allow third party contribution through open APIs and hardware interoperability. The OpenAIS architecture does not specify the used network access IP protocol suite layer and also independent of the IPv6 implementation (network stack). The requirements towards the network access layer:

- supports IPv6 multicasting,
- supports a minimum path MTU of 1280 bytes, where MTU stands for the maximum transmission unit,
- link-layer security if the communication is wireless.

The architecture also requires that the total latency between the press of a button and its respective light going on is less than 400 ms. Calculating this, delays from both the upper and network access layers need to be considered, at the latter one with aspects such as the bandwidth, packet loss rate, hops, or duty cycling.

1.1.3 OpenAIS requirements for time synchronized operations

The OpenAIS architecture specifies the requirement of synchronized and consistent operation at command level among devices in the network. There are several features mentioned in the OpenAIS architecture [3] that require some level of time synchronization:

Feature 1 Enable logging of events on multiple devices, which can be used for analysis and debugging.

Feature 2 Timestamp collected sensor data.

Feature 3 Luminaires may notify users about scheduled events, such as by dimming down at the end of a meeting.

Feature 4 Time scheduling functions are generally available in controlled and standalone installation.

Feature 5 Active, time-based lighting schedules, such as automatic brightness and colour temperature changes during the day.

Feature 6 Both repeated and event based timings; lights can be switched and dimmed according to wall-clock-entries (schedule).

Feature 7 Luminaires can be grouped and controlled in synchronicity.

However, the different features require different time resolution and accuracy. The OpenAIS architecture requires a very high, millisecond range accuracy for Feature 1, a second range accuracy for Feature 2 if the sensor data is collected for offline analyses, but a 100 millisecond accuracy if it is used in real-time algorithms. For grouped events Feature 7, the expected synchronicity is in the order of 200 milliseconds. A summary of these is depicted in Table 1.1.
Table 1.1: OpenAIS requirements for time synchronization

<table>
<thead>
<tr>
<th>Feature</th>
<th>Required accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature 1</td>
<td>millisecond</td>
</tr>
<tr>
<td>Feature 2</td>
<td>100 millisecond</td>
</tr>
<tr>
<td>Feature 3</td>
<td>second</td>
</tr>
<tr>
<td>Feature 5</td>
<td>second</td>
</tr>
<tr>
<td>Feature 6</td>
<td>second</td>
</tr>
<tr>
<td>Feature 7</td>
<td>200 millisecond</td>
</tr>
</tbody>
</table>

1.2 Problem statement and formalization

As noted by Tanenbaum et al. [5], nearly all computers have a circuitry for time keeping. This circuitry usually contains a quartz crystal, which under tension oscillates at a nominal frequency. Electronic oscillators without a crystal are used to drive low-clock processors cannot be used as reliable time keeping sources due to their low stability. Stability shows how well an oscillator can produce the same time over a given time interval. Hence, time can be measured by incrementing a counter register in the hardware after a certain number of oscillations. If we consider $t$ as the real time, a local clock $L_i(t)$ can be computed, which represents the local time of node $i$ on a network:

$$L_i(t) = \theta_i \cdot t + \phi_i$$  (1.1)

where $\theta_i$ is the drift rate, and $\phi_i$ is the offset. As noted by Karl et al. [6] influencing the oscillator is neither possible nor desirable, therefore the coefficients $\theta_i$ and $\phi_i$ can be used to adjust the local clock as explained below.

Problem 1

In a network, we will encounter different start-up times between nodes. These start-up times will result in $\phi$ offsets (or phase shifts) between the node clocks.

Problem 2

Crystal oscillators have a random deviation from their nominal frequency due to crystal imperfections. The accuracy of the crystal is given in parts per million (ppm), expressing the deviation from the nominal oscillation frequency.

The crystal frequency is also affected by environmental conditions such as changes in temperature, pressure, or voltage. The crystals also experience a gradual change of frequency over time (aging). These can be expressed with the oscillator’s $\theta$ drift rate.

Using equation 1.1 we can compare the relation between local clocks of two nodes on a network as:

$$L_i(t) = \theta_{i,j} \cdot L_j(t) + \phi_{i,j}$$  (1.2)

where $\theta_{i,j}$ is the relative drift rate between nodes $i$ and $j$, $\phi_{i,j}$ is the relative phase shift between nodes $i$ and $j$. If node $i$ and $j$ are perfectly synchronized, their relative drift $\theta_{i,j}$ is 1 and their relative offset $\phi_{i,j}$ is 0.

Karl et al. [6] defines the accuracy of clocks within a network for two distinguished cases:

External synchronization

Nodes are said to be accurate at time $t$ within a bound $\delta$ if

$$\forall i : |L_i(t) - t| < \delta.$$  (1.3)

Internal synchronization

Nodes are said to agree on the time with a bound of $\delta$ if

$$\forall i, j : |L_i(t) - L_j(t)| < \delta.$$  (1.4)

If nodes are externally synchronized with bound $\delta$, they are also synchronized internally with bound $2\delta$.  

6 Time synchronization in IoT lighting control
To eliminate the differences in phase shifts explained in Problem 1, an initial synchronization is necessary. Problem 2 implies that $|L_i(t) - t|$ can become as much as $\theta \cdot \Delta t$ large at a $\Delta t$ time after a synchronization. To keep the clocks accurate within a bound $\delta$, periodical synchronizations with a maximum period of $\delta/\theta$ is necessary.

The features in section 1.1.3 define actuating and sensing events on a node. An event $E$ on a node $i$ can be described by

$$E(i, u, id),$$

where $u$ is the timestamp of the event registered at the local clock $L_i(t_{registered}) = u$, and $id$ is a unique event identifier. Events can be distinguished as logged events if $u \leq L_i(t)$ and scheduled commands if $u > L_i(t)$.

Feature 1 requires an analysis of logged events between the network nodes, one example of this would be to compare chronology. To compare events $E(i, u_i, id_i)$ and $E(j, u_j, id_j)$ using equation 1.2:

$$u_i = \theta_{i,j} \cdot u_j + \phi_{i,j}$$

In order to compare the timestamps, the relative drift $\theta_{i,j}$ and the relative offset $\phi_{i,j}$ have to be known. Ganeriwal et al. [7] classifies the problem into three models:

1. The values of $\theta$ and $\phi$ are acquired at a later point (posteriori). The node clocks can be compared only at that point.
2. The values of $\theta$ and $\phi$ are exchanged regularly between the nodes. The node clocks can be compared internally at any time.
3. The values of $\theta$ and $\phi$ are compared regularly to an external reference node. The node clocks can be compared both internally and externally at any time.

Logged events need to be observed only at a later point after their occurrence, thus the first – simplest – model could be applied. However, other features require synchronized execution of scheduled commands. For this, one of the more complex, second or third models must be applied.

All of these models assume that the drift rates are constant during operation. This would mean that $L_i(t) = t$ for all $i$ nodes, or in other words $dL/dt$ is constant 1. A timer can said to be working within its specification [5], if there exist a constant $\rho$ maximum drift rate (usually specified by the oscillator’s manufacturer) where

$$1 - \rho \leq \frac{dL}{dt} \leq 1 + \rho.$$ 

Both the logged events and scheduled commands require a higher priority for internal synchronization over external synchronization. However, external synchronization also ensures - a less accurate - internal synchronization. Therefore the following problems have to be considered:

1. The $\phi_{i}$ phase shift has to be eliminated for $\forall i$.
2. If $\delta$ is the required accuracy, then nodes need to be accurate at time $t$ within a bound $\delta/2$, or nodes need to agree on the time with a bound $\delta$.

For this, changes in the $\theta_i$ drift rate need to be compensated for $\forall i$.

1.3 Goals

The features in section 1.1.3 require to synchronize every object participating in the control paradigm of OpenAIS to agree on a common time. It is advantageous, if this time is the global time. The problem of time synchronization has an extent literature, but none of them comes without compromises. Different protocols may achieve a different level of accuracy, may use different amount of resources, may have a different complexity, may use different communication
CHAPTER 1. INTRODUCTION

models, may or may not require special hardware, or may maintain synchronicity with different methods. To select the most suitable protocol many requirements should be considered.

Even with the “best” protocol selected, the choices made in the architecture may have an influence. In case of OpenAIS, these choices may include the independence of the physical layer, the support for both wired and wireless networks, or the support for wireless multihop communication.

It is expected, that the achieved accuracy is dependent on efficiency. A higher accuracy may be achieved by sending network messages more frequently, by using more computing and storage resources, or adding restrictions to the used hardware.

In my research I am investigating the following questions:

Research Question 1 What are the best protocols to efficiently synchronize time in the OpenAIS architecture?

Research Question 2 What are the architectural choices in OpenAIS that may have an influence on the protocol(s)?

Research Question 3 What accuracy can be achieved with the chosen protocol(s) when applied in the OpenAIS architecture?

1.4 Outline

This first chapter introduces the OpenAIS architecture and its requirements for time synchronization. The problem of time synchronization is formally described and the motivations for this research are defined.

Chapter 2 presents a survey of existing research on time synchronization. It points out the importance of accurate timestamping and the main difference between synchronizing on lower network layers versus on top of IP. Metrics are defined for comparison and for later evaluation.

Chapter 3 presents the NTP protocol and its implemented lightweight version, SNTP. The chapter presents the operating systems that are used in the OpenAIS pilot implementation, Mbed 3 and Mbed 5.

Chapter 4 presents the Mesh Time Synchronization protocol that is inspired by the 6LNTP protocol of Hong et al. The IEEE 802.15.4 and 6LoWPAN network standards are introduced. As they are used in the OpenAIS pilot implementation to connect devices through a wireless mesh network, they are selected for an evaluation platform of this research as well.

Chapter 5 presents the evaluation of the proposed time synchronization protocols both over the wired Ethernet and the wireless 6LoWPAN. The evaluation includes clock drift, synchronization accuracy, scalability and security considerations.

Final conclusions and future work are presented in Chapter 6.
Chapter 2

Related work

2.1 Time and time standards

Time is one of the seven base quantities defined in the International System of Units (SI). The definition of unit is the second, defined as the duration of 9 192 631 770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the caesium-133 atom. [9]

A time standard is a specification for measuring time. This can include the rate as time passes (e.g. the SI second), specific points in time (e.g. AD 2016), or both. The time standard used in this research is UTC, but a handful of other time standards need to be also explained for comprehension. [10]

- Universal Time (UT or UT1) is a time standard based on Earth’s rotation. It is proportional to the rotation angle of the Earth with respect to distant quasars. [11]

- International Atomic Time (TAI) is based on the notional passage of the SI second. TAI is kept by the International Bureau of Weights and Measures (BIPM) [12], and is based on the combined input of many atomic clocks around the world, each corrected for environmental and relativistic effects.

- Coordinated Universal Time (UTC) is an atomic time scale based on TAI, but designed to approximate UT1. UTC differs from TAI by an integral number of seconds. Sometimes leap seconds are added to keep it within 0.9 second of UT1.

- Standard time in a region deviates a fixed, round amount, usually a whole number of hours, from UTC. The offset is chosen such that a new day starts approximately while the sun is crossing the nadir meridian. In some cases the difference is not really fixed, but it changes twice a year a round amount, usually one hour (daylight saving time).

Although TAI is kept by the BIPM, it is not distributed directly, instead UTC is the primary standard used around the world to regulate clocks. National time-service laboratories maintain an approximation of UTC [12] and these approximations may be disseminated real-time via radio or other services.

The global navigation satellite systems – currently GPS and GLONASS – equipped with their own atomic clocks also disseminate the UTC time.

BIPM compares these approximations regularly and publishes the offsets for correction monthly. Receivers are available for the aforementioned time-dissemination services, but it is not always possible to equip every device with them because of the cost implications. The most common solution today is to equip one device with a receiver and connect it with the other devices on a common network. In order to synchronize time over this common network, the primary time holder needs to be able to transmit its time, and the other devices need to be able to receive that
and adjust their clocks as required. There are various distributed time synchronization protocols available, discussed in the following section.

### 2.2 Network time synchronization protocols

#### 2.2.1 Delay components in a message

If the primary time holder wants to transmit its time it has to be able to capture an internal time value at a well-defined moment. This is referred to as timestamping and is a fundamental step of every synchronization protocol. Sending that timestamp over a data link introduces a delay with constructing, transmitting and processing the message. This delay can be broken down into different components [13][14][15]; for choosing the ideal moment of timestamping, these components have to be considered.

**Send time** Various operating system delays (such as context switches) after constructing the message.

**Access time** The time the message spends waiting in the network interface. Usually the most significant component; depends highly on the used MAC protocol.

**Transmission time** Refers to the delay between the first bit and the last bit leaves the interface.

**Propagation time** The propagation between the network interfaces. At direct communication it can be neglected; over a complicated network structure or multi-hop communication it can be the most significant component in the delay time.

**Reception time** Refers to the delay between the first bit and the last bit arrives at the interface. If the message didn’t change medium, then it equals to the transmission time.

**Receive time** The time the message spends waiting in the network interface. Processing of the message will be pending until the host is ready.

![Delay components between constructing the message on the source and processing it on the destination](image)

Figure 2.1: Delay components between constructing the message on the source and processing it on the destination.

Figure 2.1 shows these delay components between two communicating hosts. If a protocol captures a timestamp on the MAC-layer, only the transmission time, the propagation time and the reception time will be added to the delay. If a protocol captures the timestamp on the application layer (in the “Time Application”), the delay will also include the send time and the receive time.

#### 2.2.2 Time protocol

The Time Protocol is defined in RFC 868 [16] by Postel and Harrenstein. The protocol is a simple time synchronization method: a client requests the time from a time server over TCP/IP.
CHAPTER 2. RELATED WORK

or UDP/IP, to which the server replies with its current time. The protocol has only a second-precision and does not compensate for possible network delays.

2.2.3 Cristian’s algorithm

In the synchronization scheme Remote Clock Reading proposed by Cristian [17] – similarly to the Time Protocol – clients synchronize to a time server. The client measures the round-trip time of the message exchange and sets its own time to the received time compensated by the half of the round-trip. Cristian argues that this method is only satisfying if the required accuracy is shorter than the round-trip time. To mitigate random delays occurring in the network, Cristian proposes to repeat the message exchange several times and use the one with the smallest round-trip delay.

2.2.4 Network Time Protocol

The Network Time Protocol (NTP) invented by Mills is the most widely adopted time synchronization protocol. NTP is also an established Internet Standard protocol, currently at its 4th version (NTPv4) as documented in RFC 5905 [18]. NTP operates on UDP/IP with a two-way message exchange. The protocol also describes an architecture that ensures the reliable and secure time synchronization over the internet. NTP is able to synchronize time with an accuracy of tens of milliseconds over the internet, and in the order of 100 microseconds in local networks. [19]

The Simple Network Time Protocol (SNTP) is an implementation of NTP without the complicated mathematical, statistical methods and clock adjustment techniques that are normally included in the full protocol. SNTP is ideal for clients with limited resources, or for systems where reliability is less important.

2.2.5 Precision Time Protocol

The IEEE 1588 precision time protocol (PTP) [20] provides synchronization with a sub-microsecond precision. The protocol is optimized for the minimal use of network bandwidth and low processing overhead. The protocol is designed for applications where a higher accuracy is required than that can be achieved with NTP, but a UTC reference clock (such as GPS) cannot be used on every host. As Mills explains [21], PTP is most commonly used in special-purpose networks, where devices are connected to a dedicated, high-speed Ethernet interconnected by switches. NTP on the other hand is most commonly used to synchronize times on machines connected over the internet by routers and gateways with a varying network speed.

At system initialization the protocol uses the Best Master Clock algorithm to automatically determine which clock in the network is the most precise, which then becomes the master clock. All other clocks become slaves and synchronize their clocks with the master. The master broadcasts its current time at a predefined period in a “Sync” and then a correction in a “Follow-up” message. If the master’s hardware allows accurate hardware timestamping, the Follow-up messages are omitted. The slaves are able to compensate for network variation as they wait for a second round of “Sync” and “Follow-up” message pair. As a next step each slave needs to calculate the Slave-Master delay by individually sending a “Delay request” message to the master. After receiving the timestamped “Delay Response” message, the slave can average the two – Master-Slave and Slave-Master – delays and adjust the time of its clock.

The 2008 revision of the standard, IEEE 1588-2008, introduces a protocol for network equipments (switches, routers), that corrects the timestamps in the PTP messages with the time spent traversing the equipment. This compensates the in-determinism across the network.

2.2.6 6LNTP

The aforementioned protocols were designed for Ethernet, where the access time (as defined in section 2.2.1) can be neglected when using a modern network built with switches and full-duplex
connections. However, in case of a wireless communication, where special medium access protocols are used, the access time can be significant and highly varying. If we consider multihop mesh networks, the access time and its uncertainty accumulates, thus the common network delay estimation methods fail. A protocol called 6LoWPAN Network Time Protocol (6LNTP) was proposed by Hong et al. \cite{8} targeting time synchronization over 6LoWPAN multihop mesh networks. The protocol has two phases:

**Gateway phase** The 6LoWPAN gateway corrects its time by exchanging time request and response packets with an NTP server.

**LoWPAN synchronization phase** The wireless nodes synchronize their time with the gateway. In return to a node’s time request, the gateway sends out a synchronization and a follow-up multicast message with a link-local scope. Multicast on the link-local scope in 6LoWPAN is essentially a link-layer broadcast and is not retransmitted. The sync message includes a reference time, while the follow-up message includes a time correction calculated by the sum of the send time, access time and transmission time elapsed since the reference timestamp. The intermediate nodes receive the sync message, they store the reference time and rebroadcast it to their own neighbors. When they receive the follow-up message, they set the time on their clock and also re-broadcast this message, but corrected with the errors introduced during the retransmission. As Hong et al. explains, the benefit of 6LNTP is that intermediate nodes are also synchronized when another node on the mesh network requests time synchronization. This reduces the amount of message exchange compared to a scenario where the nodes would individually synchronize with unicast messages. \cite{8}

Hong's evaluation shows an average synchronization error of 542.875 $\mu$s, 593.636 $\mu$s, and 788.246 $\mu$s with standard deviation of 0.542, 0.585, and 0.786 over 1-hop, 2-hop, and 3-hop communication respectively.

### 2.2.7 Synchronization protocols for wireless sensor networks

The topic of time synchronization in sensor networks provided an inexhaustible field for research in the recent years. Sensor networks have widely varying requirements for energy efficiency, robustness, scalability, cost and size, internal or external synchronization, or resource constraints. They are usually designed for data collection, often with either sporadic or well-scheduled network communication, thus making timestamping and/or time synchronization crucial on them. However it is important to mention, that these synchronization methods commonly accomplish only an internal synchronization. They are also characterized by a homogeneous hardware setup, that allows the creation of specialized algorithms that are based on that specific hardware and not on a standardized protocol.

The previously presented synchronization methods often have requirements for properties such as error-free operation, a large storage for data sampling, or higher processing power – for properties that are usually the first to be sacrificed when designing wireless sensor networks. Therefore different synchronization schemes can be found in the open literature \cite{14} to provide some level of time synchronization by overcoming one or more of the constraints:

- Tiny-Sync and Mini-Sync proposed by Sichitiu and Veerarittiphan are focusing on low-complexity. \cite{14}
- Lightweight Tree-based Synchronization (LTS), proposed by Greunen and Rabaey aims at minimizing the complexity of the synchronization (such as the exchanged messages) for the price of reduced accuracy. \cite{14}
- Mirabella et al. \cite{22} proposes the Dynamic Continuous clock Synchronization scheme to maximize sleeping time by introducing a virtual clock for TDMA based MAC protocols.
- Lee and Choi \cite{23} proposes the Chaining Clock Synchronization (CCS) to maximize energy-efficiency by sacrificing accuracy.
• In the Reference Broadcast Synchronization (RBS) \cite{15} proposed by Elson, Girod and Estrin, the nodes don’t synchronize to a select reference node, but with one another by sending one-way reference beacons to their neighbors. The reference beacons’ time of arrival is used by the receiving nodes for comparing their clocks. This way \( E(i, u, id) \) events can be compared later after their occurrence by extrapolating backwards to estimate the offset between clocks. The authors argue that, by removing the role of reference node, the only source of error will be the non-determinism of propagation time and receive time.

The authors achieve an 11\( \mu \text{s} \) accuracy in their evaluation on Berkeley Motes, a widely used sensor node.

• PalChaudhuri et al. \cite{24} and Cao et al. \cite{25} propose an internal synchronization scheme for multihop networks as an RBS extension by sacrificing accuracy.

• The Timing-Sync Protocol for Sensor Networks (TPSN) proposed by Ganeriwal et al. \cite{7} has two phases. In the first, Level Discovery Phase, a so called root node is assigned level 0; this can be assigned externally, or nodes can take over periodically. The root node initiates the level discovery phase, where each node on the network is assigned a level to create a hierarchical topology through network flooding. The assigned levels essentially represent the hop-distance of the node from the root node.

The second, Synchronization Phase initiated by the root node in several rounds where the nodes reach a level-by-level synchronization starting from the lowest level. By the end of the phase each node is synchronized to the root node as well.

TPSN handles node failures, newly joined nodes, and the failure of the root node. The authors argue that TPSN gives a 2 times better performance than RBS implemented on the same Berkeley Motes, although they report 16.9\( \mu \text{s} \) accuracy.

• The Flooding Time Synchronization Protocol (FTSP) proposed by Maróti et al. combines the strengths of RBS and TPSN promising a microsecond accuracy, but relies on a non-standardized MAC layer functionality. \cite{13}

2.2.8 Time synchronization protocol for OpenAIS

The investigated protocols are summarized in Table 2.1 with the following comparative aspects (some of them from Aoun \cite{13}):

• Accuracy: as explained in section 1.2, the accuracy of the synchronization can be expressed as the difference between the reference clock value and the node’s clock value. The achieved accuracy can highly vary between protocols.

• Communication model: synchronization messages can be broadcast, multicast, unicast, or a combination of these.

• Clock correction type: can be divided into protocols that assume a constant drift rate (offset correction only) and into protocols that compensate for the drift rate in-between synchronization rounds (both rate and offset correction).

• Synchronization maintenance: a difference here is made between proactive protocols that initiate synchronization rounds periodically, and reactive protocols that synchronize clocks in a post-facto manner, on-demand, eg. only then, when the timestamp is actually observed.

• As explained in section 2.2.1 the moment of timestamping is a fundamental question for every time protocol. Timestamping on a lower layer than where the time application runs might require support from the network stack or network interface.
The Time Protocol and Cristian’s Remote Clock Reading protocol were superseded by newer protocols that also correct for network delays and thus achieve a higher accuracy. However they are included in this survey as they serve as an example for the simplest time distribution protocols.

NTP is the most commonly used protocol to synchronize computers over the internet and as a result of its advanced algorithms it can achieve a good accuracy. Although PTP can outperform NTP on Ethernet, it requires hardware assisted time stamping to achieve this. PTP also lacks the capability to synchronize over the internet.

The 6LNTP synchronization scheme is only an extension of NTP to 6LoWPAN and mesh networks, where the wired protocol would not perform well.

The time synchronization protocols for sensor networks can achieve high accuracy, but mostly only focus on an internal time synchronization achieved between neighbors. However, our goals go beyond the extent of peer-to-peer synchronization, to acquire global, UTC-synchronized time. These protocols also assume a close coupling of software and hardware (eg. timestamping on the MAC layer to save energy), or a network composed of nodes with the similar hardware (eg. their communication on the physical and link layer).

The OpenAIS project aims at an IP based architecture where devices from different vendors can be transparently connected through and open IP interface regardless of the underlying physical layer. This implies the use of a time synchronization protocol on top of IP. The Internet of Things also assumes a consistency and precision of information on a global level, a break-out from the isolated local networks. If a personal computer connected to the internet is able to synchronize its time without further configuration, an IoT device should be able to do that too – using (mainly) the internet as its infrastructure – let it be a smart luminaire or a wall switch. From the time synchronization protocols available today, only NTP can provide a global and pervasive time synchronization over the internet. Although NTP uses resource-demanding mathematical and statistical algorithms, it was successfully applied in the domain of constrained embedded devices by using its lightweight version, SNTP. The millisecond accuracy promised by NTP also aligns with the requirements in section 1.1.3. In this research the applicability of the NTP and SNTP protocols in the OpenAIS architecture is further studied.

The protocol can be expected to perform with a lower accuracy on mesh networks: the uni-

---

Table 2.1: Comparing network time synchronization protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Accuracy</th>
<th>Communication</th>
<th>Clock correction</th>
<th>Sync maintenance</th>
<th>Timestamping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time protocol</td>
<td>Compares to the round-trip time</td>
<td>Unicast</td>
<td>Offset correction</td>
<td>Proactive</td>
<td>Application layer</td>
</tr>
<tr>
<td>Cristian’s algorithm</td>
<td>Compares to the round-trip time</td>
<td>Unicast</td>
<td>Offset correction</td>
<td>Proactive</td>
<td>Application layer</td>
</tr>
<tr>
<td>NTP</td>
<td>Milliseconds</td>
<td>Unicast or multicast</td>
<td>Drift and offset correction</td>
<td>Proactive</td>
<td>Application layer</td>
</tr>
<tr>
<td>PTP</td>
<td>100 nanoseconds</td>
<td>Multicast</td>
<td>Offset correction</td>
<td>Proactive</td>
<td>Appl. layer, with HW provisions</td>
</tr>
<tr>
<td>6LNTP</td>
<td>Milliseconds</td>
<td>Unicast and broadcast</td>
<td>Offset correction</td>
<td>Proactive</td>
<td>Application layer</td>
</tr>
<tr>
<td>Sensor networks</td>
<td>Microseconds</td>
<td>Broadcast</td>
<td>Drift and offset correction</td>
<td>On-demand or Proactive</td>
<td>MAC layer</td>
</tr>
</tbody>
</table>
cast/multicast communication model does not scale well over multiple hops and the used medium access control protocol in shared-medium wireless communication may have an impact on the compensation method in NTP. Therefore, further research is done on these effects and complementing NTP with the 6LNTP scheme.

2.3 Clock drift correction techniques

Besides NTP and some sensor network time synchronization protocols, the described protocols only correct the clock offsets. As discussed in section 1.2, to keep clocks accurate over time, the drift rate $\theta$ of every node need to be compensated. The crystal-inherited problems are discussed in this section, excluding problems originating in the oscillator-electronics (such as mismatching load capacitance or voltage stability).

**Frequency tolerance** Crystal oscillators have a random deviation from their nominal frequency. This is mainly due to imperfections in the crystal, hence no two crystals the same. The deviation from the nominal oscillation frequency – also referred to as frequency tolerance – is given in parts per million (ppm). The typical deviation of low-cost 32.768 kHz crystals found in most of the embedded systems is $\pm 20$ ppm. This equals to $\pm 20$ microsecond deviation in a second. The error originating from frequency tolerance can be compensated by accurately measuring the crystal frequency, however this is often not possible due to its cost implications. NTP includes complicated mathematical, statistical methods to calculate the necessary drift compensation. If these calculations are not implemented, the synchronization round must be repeated regularly.

**Frequency stability** The crystal frequency is also affected by environmental conditions such as changes in pressure, voltage, or temperature. For OpenAIS, the most important condition to be considered is temperature, as the nodes will operate in various office conditions: from normal room temperature, to very high (100 $^\circ$C) temperatures experienced in luminaries. This also means changing environmental conditions, considering luminaries turned on or off. The oscillator frequency has a negative parabolic dependence over temperature. This is specific to the crystal and it can be found in the manufacturer’s specification. A typical 32.768 kHz crystal’s frequency-temperature characteristics is shown on Figure 2.2. As it can be seen, not even a constant compensation would bring improvement, since the crystal will introduce more error at lower and higher temperatures.

![Figure 2.2: ABRACON ABS07-32.768KHZ-T crystal frequency-temperature characteristics. Taken from: [26]](image-url)
CHAPTER 2. RELATED WORK

Some temperature-drift pairs are collected from Figure 2.2 and depicted in Figure 2.3 with the elapsed time after an initial synchronization in minutes on the horizontal axis, versus the typical drift in milliseconds on the vertical axis.

![Figure 2.3: Absolute crystal drift including random errors at various temperature conditions](image)

By measuring crystal temperature, it is possible to correct the temperature-implied drift in software with an opposite positive parabola. One method is to store the crystal temperature dependence as a lookup table and correct the time measurements. Another method is to obtain a curve-fitting calculation with coefficients calculated from the parabola. As measuring the crystal temperature is often not possible, the assumption is made that the crystal follows the board-temperature [27]. This is valid, if the on-board parts (e.g., voltage regulators) do not create temperature gradients on the PCB. If the board-temperature cannot be acquired either, then the assumption can be made that the crystal follows the on-die temperature of the micro-controller. In this case the processing activity should be reduced to minimum to avoid undesired rise in the on-die temperature.

**Aging** The crystals also experience a gradual change of frequency over time, called aging. Typical crystals [27] experience a change up to $\pm 3$ ppm in the first year. The crystal manufacturer usually specifies the first year aging, but this does not mean the aging stops after this point. Aging also affects other oscillator components, such as capacitor, but their effect on the oscillator is negligible ($\approx 0.1$ ppm).

Drifts due to aging can be corrected by calculating the long-term average drift: by timestamping at the time the clock is synchronized, the drift difference can be calculated at the next synchronization. To exclude short-term drift (random deviation and environmental conditions), the calculated difference should be averaged over a longer period of time before using it as a compensation factor. This may require available static memory area on the node.

Timekeeping in embedded systems is often assigned to a real-time clock (RTC). The RTC function can be implemented either as a separate system, or as an integrated circuit together with a processor. The RTC benefits the system with low power consumption, higher timekeeping accuracy, and improved reliability against system lockup. As manufacturer notes describe [27], using a combination of the above mentioned techniques, the drift rate can be calibrated and compensated to an acceptable accuracy ($\pm 20$ ppm). In this research, however, a periodically scheduled synchronization is used.

### 2.4 Performance metrics for evaluation

In order to evaluate the performance of the selected time synchronization protocols, some performance metrics has to be decided on. The following key metrics come from the OpenAIS use
cases and requirements:

**Accuracy** The most important requirement for a time synchronization protocol. In the followings, accuracy is expressed as the clock offset w.r.t. the time reference.

**Scalability** As Tanenbaum mentions [5], scalability is one of the most important design goals in distributed systems. An Internet of Things system has to be able to accommodate future expansions as the number of devices, resources can grow and underlying physical layer can change. The most important aspect of scalability for a network time synchronization protocol is the network traffic it generates. OpenAIS supports both wired and wireless mesh technologies. A lighting application may be expected to generate light traffic, the state-of-the-art Ethernet network should handle that without difficulties, however, in the wireless mesh designed for constrained, low-power devices bandwidth is limited.

**Security** A time service can be vulnerable to attacks that might attempt to disrupt the service or corrupt the collected data. Security is the main concern in the spread of IoT and for reference designs such as OpenAIS, it is a key to think of security in their foundations. Options for securing the time synchronization need to be considered.

Chapter 5 further explains how these metrics are measured.
3.1 Network Time Protocol

Mills’ NTP protocol is an established Internet Standard protocol [18], currently at its fourth version. NTP is used to synchronize computers over the internet to UTC.

3.1.1 NTP timestamp format

NTP defines three time formats: a 128-bit date format, a 64-bit timestamp format, and a 32-bit short format. The 128-bit format is used where sufficient storage and word size are available. The 64-bit timestamp format is used in NTP packet headers. It includes a 32-bit unsigned seconds field enough to span 136 years and a 32-bit fraction field enough for a resolution of 2^32 picoseconds (Figure 3.1). The short format is used where the range and resolution of other formats are not necessary. All values are represented in twos-complement format, in big-endian order.

<table>
<thead>
<tr>
<th>LSB</th>
<th>MSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds (32 bit)</td>
<td></td>
</tr>
<tr>
<td>Fraction (32 bit)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.1: NTP Timestamp format

The prime epoch, or base of the date and timestamp formats is 0 h 1 January 1900 UTC. Era 0 starts from this epoch and lasts till some time in 2036, when the timestamp field wraps around and Era 1 follows. Although the timestamp format does not include an era number, an $L_i$ node’s clock can be successfully synchronized to a $t$ clock, if it is initially set to a value where it is true that $|L_i(t) - t| < 68$ years.

3.1.2 NTP message format

An NTP packet is a UDP datagram [28]. It consist of a header, an optional extension field, and an optional message authentication code. The latter two may be used by cryptographic algorithms. The header size is 48 bytes and it contains the following variables:

**Leap Indicator** As described in section 2.1, leap seconds are occasionally applied to UTC in order to keep it close to UT1. The Leap Indicator is a 2-bit integer warning of an impending leap second in the last minute of the current month.
CHAPTER 3. SNTP

**Version Number** A 3-bit integer representing the version number. The current version number is 4.

**Mode** A 3-bit integer representing mode, such as 3 for client, 4 for server and 5 for broadcast mode.

**Stratum** An 8-bit integer representing the stratum number. A stratum value 16 means an unsynchronized server.

**Poll** An 8-bit integer representing the maximum interval between successive messages, in \( \log_2 \) seconds.

**Precision** An 8-bit signed integer representing the precision of the system clock, in \( \log_2 \) seconds.

**Root delay** The total round-trip delay to the reference clock expressed in NTP short format.

**Root Dispersion** The dispersion represents the maximum error inherent in the measurement in NTP short format. The root dispersion is the accumulated error to the reference clock.

**Reference ID** For stratum 1 reference, the reference ID is a string, such as “GPS”. Above stratum 1 – in case of using IPv6 – it is a hash of the IP address.

**Reference Timestamp** The time represented in NTP timestamp format, when the system clock was last set or corrected.

**Origin Timestamp** The client’s time at the time request packet transmission represented in NTP timestamp format.

**Receive Timestamp** The server’s time at the request packet reception represented in NTP timestamp format.

**Transmit Timestamp** The server’s time at the response packet transmission represented in NTP timestamp format.

### 3.1.3 On-wire protocol

The most important mechanism in the protocol – the on-wire protocol – is used to exchange time values between nodes. It follows a client-server model, where the server represents the time reference, and the client represents the host to be synchronized.

The request message sent by the client contains the timestamp of the packet transmission (origin timestamp – \( t_0 \)). The server receives the packet and timestamps the reception (receive timestamp – \( t_1 \)). When the server responds (transmit timestamp – \( t_2 \)), it includes all these timestamps \( (t_0, t_1, t_2) \) in the packet. The client then timestamps the reception of the response packet (destination timestamp – \( t_3 \)). Note, that this last timestamp is never included in the NTP packet, as it is determined on the client. Timestamps \( t_0, t_3 \) are taken with the local clock of the client, while timestamp \( t_1, t_2 \) taken with the local clock of the server. An example for the message exchange is depicted in Figure 3.2. Using the four timestamps the client can calculate its \( \theta \) offset and \( \delta \) round-trip delay as the following:

\[
\theta = \frac{(t_1 - t_0) + (t_2 - t_3)}{2}
\]

\[
\delta = (t_3 - t_0) - (t_2 - t_1)
\]

On Figure 3.2, after a successful message exchange the client can calculate its offset \( \theta \) as the following: \( \theta = \frac{((t_1 - t_0) + (t_2 - t_3))/2 = ((135ms - 231ms) + (137ms - 298ms))/2 = -128.5ms \). The \( \delta \) round-trip delay marked with blue can be calculated as \( \delta = (298ms - 231ms) - (137ms - 135ms) = 65ms \). It’s important to note, that the calculations assume a symmetric delay on the channel.
CHAPTER 3. SNTP

The message exchange protocol also has a “broadcast” variant, where the server periodically sends multicast messages that can be received by multiple clients. In this mode the clients still have to do several exchanges with the client-server mode to calculate the message delay. When receiving the multicast messages, this calculated message delay is used.

3.1.4 Hierarchical timeflow

NTP is designed to automatically organize nodes into a hierarchy that ensures that the synchronization network does always produce the most accurate time available and it is reliable in case of failures.

The hierarchy, depicted in Figure 3.3, has a layered tree architecture, where each layer is defined with a stratum number. On stratum 1, the so called primary time servers synchronize directly to a UTC reference source (stratum 0), such as an atomic clock, or GPS. A server synchronized to a stratum n server will become a stratum n + 1 time server. The highest stratum level is 15.

The time flows from the primary servers through the secondary servers to the client leaves. As Mills explains [18], mean errors, measured by synchronization distances, increase approximately in proportion to stratum numbers and measured round-trip delay. Hence, the topology is organized with a routing algorithm to minimize the synchronization distance and avoid loops.

![Figure 3.2: The NTP algorithm](image)

![Figure 3.3: The NTP architecture](image)
CHAPTER 3. SNTP

The hierarchy also can be seen as a master-slave network, where the primary servers are masters, however, there is no need for an election protocol. As it is depicted in Figure 3.3, the nodes also exchange synchronization messages with servers on the same stratum. This peer-process enables sanity checking and backup on stratum 1, while on the higher layers it improves reliability and provides a more stable time as the peers slowly converge towards a common time. The peer process is using the same on-wire protocol for communication described in section 3.1.3, but it is managed by poll process, selection, clustering, and mitigation algorithms.

3.1.5 Clock adjustment and discipline processes

As discussed in section 1.2, time synchronization is a two-fold problem: the elimination of offset $\phi$ and compensating drift rate $\theta$. NTP’s goal to minimize both between UTC and the node’s clock.

NTP defines various mitigation algorithms. The $\theta$ and $\delta$ values acquired using the on-wire protocol are used in statistical algorithms: outliers are discarded and an estimate offset is derived from the best remaining candidates. NTP’s clock discipline and clock adjustment processes then act as a variable frequency oscillator: it creates a feedback loop to adjusts the clock frequency, thus reducing the offset gradually. These mitigation and discipline algorithms extend the potential accuracy to tens of microseconds.

3.2 NTP for constrained devices

3.2.1 SNTP

A subset of NTP, lacking the mitigation algorithms mentioned in section 3.1.5, called the Simple Network Time Protocol is also defined in [18] and [29].

An SNTP server implementation is intended for primary servers equipped with a reference clock. An SNTP client implementation is intended for clients with only one upstream server and no dependent clients. The NTP and SNTP packet formats are the same, and the arithmetic operations to calculate the client’s clock offset and round-trip delay are the same [18] [29]. Hence NTP and SNTP servers and clients are interoperable and can be internixed. SNTP servers and clients, however, should be placed on the leaf layers of the architecture.

SNTP is popular in constrained embedded devices as it does not require the implementation of the complicated, resource-hungry statistical algorithms, or storing states over time. On the other side, the lack of these also reduces accuracy and reliability. The simplest SNTP client implementation may use only the transmit timestamp field of the server packet to set its time. As the additional complexity to implement the full on-wire protocol is minimal, Mills encourages its implementation.

3.2.2 Software architecture

The OpenAIS project also prepares a pilot implementation and system design. The foundation of the software architecture is built upon ARM Mbed. The ARM Mbed [30] is a framework designed for the Internet of Things. The framework consist of

- mbed devices, whose basic hardware is the development board including an ARM microcontroller and other additional components (sensors, actuators, storage extensions etc.)
- mbed OS, a new operating system for mbed devices;
- mbed Device Server (mDS), a middleware for managing and connecting mbed devices;
- mbed Tools for building, packaging and testing the code for mbed devices.
CHAPTER 3. Sntp

Mbed OS 3

In April 2015 ARM announced the Mbed OS 3 [31]. It is an open source operating system designed specifically for constrained devices. It allows C++ applications to run on these devices; provides hardware abstraction APIs to control them and run the same code on any supported board. Mbed OS 3 is single-threaded and event-driven, responsible for task scheduling and power management. The system is designed to be modular; the modules are only compiled if the application needs them, thus saving device resources.

Event scheduling in Mbed 3

MINAR [32] is mbed OS’s task scheduler. The act of scheduling to run a task is called posting a callback. A callback can be posted to run after a timeout or periodically with, or without a time tolerance. Tasks will never interrupt each other and will always run to completion before giving back the control to MINAR. The scheduler aims to maximise processor sleep time considering the posted task execution times and tolerances. MINAR is neither a pre-emptive, nor a real-time scheduler, hence task execution time has to be kept as short as possible (non-blocking). This model may have an impact of the accuracy of time synchronization: as events do not interrupt each other, the moment of timestamping will be delayed by the execution time of the tasks already present in the event-queue.

Mbed OS 5

In August 2016 Mbed OS 3 was superseded by Mbed OS 5 [33]. Mbed 5 provides native thread support by incorporating a real-time operating system based on the open-source CMSIS-RTOS RTX in its core. This enables both blocking and non-blocking design patterns, and a deterministic, multithreaded real time software execution. Events may also happen in an interrupt context, thus applications rely on features such as semaphores, mutexes and threads.

This research contains findings for time synchronization implemented both on Mbed OS 3 and Mbed OS 5. The evaluation in Chapter 5 gives a comparison of their effects on the time synchronization.

Ticker APIs

Both Mbed OS 3 and 5 offer two different timers – or so called tickers – through their HAL module [34]. The HAL module defines the API for the low level hardware abstraction layer, accessing the hardware modules such as GPIOs, GPIO IRQs, I2C, SPI, PWM out, etc. The implementation of these APIs are built from target-specific submodules (specific to the vendor and chip), but their standard interface provides the basis for the upper-layer system modules (networking stacks, schedulers etc.). The following observations were made for the FRDM-K64F board (see later in section 5.1.1) specifically:

lp ticker: a low power, millisecond precision timer. It is implemented with using the real-time clock (RTC) on the board. MINAR in Mbed 3 uses the lp ticker for the scheduling of tasks.

us ticker: a microsecond precision timer. It is implemented with using a Programmable Interval Timer (PIT) on the board.

The HAL module also provides an API to read and write the RTC; but it is discouraged to do so, as other applications – just as the lp ticker itself – may rely on its value (such as the MINAR scheduler in Mbed 3). In this implementation the us ticker was used as a primary reference for measuring time.

Time synchronization in IoT lighting control 23
3.2.3 Implementation on Mbed OS

This section shortly describes the related classes for the SNTP client implementation. The NTPTimestamp data type implements the NTP timestamp format and its allowed operations as described in [18]. The NTPPacket class is responsible for serializing and deserializing NTP messages through its member functions (mkMessage and rvMessage respectively). The SNTP class is responsible for keeping the current system time and for updating it by exchanging synchronization messages with an SNTP or NTP server. The full on-wire protocol is implemented, hence the offset \( \theta \) and the round-trip delay \( \delta \) are calculated. The interface of this class is slightly different between Mbed 3 and 5, because of differences in the network and socket interfaces on the two versions. Timestamps are captured in callbacks on Mbed 3, while on Mbed 5 directly before the message send and after the message receive operations. The class does not implement a synchronization maintenance schedule, so the main application can freely choose this. This can be as simple as scheduling synchronization rounds with a defined periodicity, or a more complex one, where the periodicity is dynamically selected (eg. based on the network communication of other applications, or when a luminaire is turned on and a higher temperature, thus a higher drift rate is expected). The wall-clock time kept by the SNTP client class can be acquired by instantiating Timestamp objects. For example, Timestamp objects are used as well to create origin \( t_0 \) and destination \( t_3 \) timestamps in the client. The classes are depicted on a UML class diagram in Figure 3.4.

![Figure 3.4: UML class diagram for the SNTP implementation](image-url)
Chapter 4

Mesh Time Synchronization Protocol

NTP is the most widely used protocol to synchronize time over the internet. The protocol, however, may perform poorly over multihop wireless networks. These types of communication methods are also supported by the OpenAIS architecture, hence the synchronization protocol proposed by Hong et al. [8] was selected to extend NTP to mesh networks. Before discussing the protocol design and implementation in sections 4.2 and 4.3, section 4.1 introduces the IEEE 802.15.4, 6LoWPAN and Thread standards used in the OpenAIS pilot implementation, representing a low-cost, low-speed, energy-efficient communication for Internet of Things devices.

4.1 Introduction

4.1.1 IEEE 802.15.4

IEEE 802.15.4 [35] is a standard maintained by a working group of the Institute of Electrical and Electronics Engineers (IEEE). The group defines standards for LR-WPANs (low-data-rate wireless personal area networks) with a focus on low-cost, low-complexity and low-power communication. The 802.15.4 standard defines the physical layer (PHY) and media access control (MAC) layer in the OSI reference model. Hence the PHY layer defines the wireless conditions of the link (such as frequency, modulation and the power) and the MAC layer defines the channel access mechanism, the data handling and the addressing.

The PHY specifications support narrowband communications in the license-free 2.4 GHz ISM band worldwide, 868 MHz band in Europe and 915 MHz band in Americas (overview in Table 4.1). Other frequency bands are also supported in China and in Japan. The OpenAIS project targets only the 2.4 GHz band. The minimum nominal transmitter power output is defined as $-3dBM$ or $0.5mW$, but the most commonly available modules are transmitting at $0dBm$ or $1mW$. The maximum available transfer rate is 250kbit/s. The maximum supported frame size is 127 bytes. The transmission range is not well-defined as it is dynamically changes based on line of sight, receiver sensitivity or transmitter power etc., but most applications consider to cover a range of 10 to 100 meters.

The 802.15.4 standard supports beacon enabled or beaconless operation. The beaconless mode uses unslotted carrier sense multiple access with collision avoidance (CSMA/CA) to access the medium, while the beacon-enabled mode a slotted CSMA/CA and a superframe structure.

The IEEE 802.15.4e [36] was introduced as an amendment for the MAC to improve robustness, latency and determinism with features such as Time Slotted Channel Hopping (TSCH), Deterministic and Synchronous Multi-Channel Extension (DSME), and Low Latency Deterministic Network (LLDN).

The standard defines two device types: a full-function device (FFD) and a reduced-function...
Table 4.1: 802.15.4 PHY layer parameters for the standard’s supported bands

<table>
<thead>
<tr>
<th>Geographical region</th>
<th>Europe</th>
<th>Americas</th>
<th>Worldwide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency band</td>
<td>868-868.6 MHz</td>
<td>902-928 MHz</td>
<td>2.4-2.4835 GHz</td>
</tr>
<tr>
<td>Channel bandwidth</td>
<td>600 kHz</td>
<td>2 MHz</td>
<td>5 MHz</td>
</tr>
<tr>
<td>Number of channels</td>
<td>1</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>Modulation scheme</td>
<td>BPSK</td>
<td>BPSK</td>
<td>QPSK</td>
</tr>
<tr>
<td>Data rate</td>
<td>20 kb/s</td>
<td>40 kb/s</td>
<td>250 kb/s</td>
</tr>
</tbody>
</table>

device (RFD). An FFD is capable of serving as PAN coordinator or coordinator, while an RFD is not, but the latter can be implemented using minimal resources and memory capacity. A WPAN includes at least one FFD, which operates as the PAN coordinator. IEEE 802.15.4 defines star and peer-to-peer network topologies (as seen on Figure 4.1). In the former, nodes in the network communicate through a central PAN coordinator node, while in the peer-to-peer topology nodes may talk to any other node. This enables upper networks layers to define mesh topologies.

IoT requires the interconnection of all the devices using Internet protocol. As seen, a 802.15.4 WPAN is characterized by small packet sizes, low bandwidth, and resource-constrained devices. The goal of the 802.15.4 standard is to provide the base to other upper layer protocols, and in order to bring IP over 802.15.4 some challenges has to be tackled by these upper layers:

**Header size** The maximum frame size (or MTU) in 802.15.4 is 127 bytes. The IPv6 header weighs in with 40 bytes, that may be an inefficient use of the limited bandwidth.

**Packet fragmentation** The minimum MTU required by IPv6 is 1280 bytes, which means some packets may be too large compared to the MTU of 802.15.4 networks.

**IP routing** The problem of routing may be twofold: messages may need to be routed in a 802.15.4 mesh on the link-layer (mesh-under), or using IP on the network layer (mesh-over).

### 4.1.2 6LoWPAN

The IETF IPv6 over Low power WPAN (6LoWPAN) working group developed standards to enable the efficient transmit of IPv6 datagrams over IEEE 802.15.4 networks [37]. The main features of the 6LowPAN protocol are the following:

**IPv6 header compression** The RFC6282 [38] standard defines an encoding format, for effective compression of Unique Local, Global, and multicast IPv6. The standard also defines a compression mechanism for UDP and encoding formats for IPv6 extension headers.

**IPv6 fragmentation** The RFC4944 [37] standard defines the fragmentation of IPv6 packets that are too large to fit into a single 802.15.4 frame. If a fragment is lost, the whole set need to be resend.

![Figure 4.1: IEEE 802.15.4 star and peer-to-peer network topologies](image-url)
IPv6 Neighbor Discovery The RFC6775 [39] standard defines an efficient method for IPv6 Neighbor Discovery (ND) and duplicate address detection. A 6LoWPAN network can have 4 different device types that allow the formation of star and mesh network topologies:

- A 6LoWPAN Border Router is the coordinator of the 6LoWPAN network and responsible for IPv6 prefix propagation in it.
- A 6LoWPAN Router is a node that can route packets.
- A 6LoWPAN Host is a node that does not route any packets.
- A 6LoWPAN Sleepy Host is a host that is allowed to sleep and turn of its radio.

Important to note that 6LoWPAN supports multicast, but the routing mechanism is expected to be specified by other mesh routing mechanisms; such protocol is the Multicast Protocol for Low-Power and Lossy Networks (MPL) defined in RFC7731 [40]. IPv6 multicast messages need to be carried by 802.15.4 MAC broadcasts.

4.1.3 Thread

![Thread stack](image)

Figure 4.2: Overview of the Thread stack. Taken from: [41]

As seen, 6LoWPAN does not define authentication methods for joining nodes, neither the routing on the network. The Thread stack [41] is an open standard on top of 802.15.4 and 6LoWPAN, to specify these with the following characteristics:

No single point of failure Thread is designed to provide a reliable communication even if individual devices fail. Thread defines Border Router, Router, Router-eligible End Device, and Sleepy End Device device types. A network Leader role elected from the Border Router and Router devices on the network, and automatically reassigned if the Leader fails.

Advanced routing While DHCPv6 is used to allocate addresses to Routers, a short address is allocated to End devices connecting to those Routers. Thread typically allows up to 32 routing nodes in a network with the possibility to address thousands of end nodes connecting to them. The current number of supported devices in a Thread mesh is 250. Routers exchange their cost of routing with each other in a compressed format, using the Mesh Link Establishment (MLE). Distance vector routing is used to keep routes to Router addresses. For routing outside the network, the Border Router(s) notify the Leader about their prefixes, which are then also distributed in the mesh with MLE.

Time synchronization in IoT lighting control 27
CHAPTER 4. MESH TIME SYNCHRONIZATION PROTOCOL

Network authentication Devices need to be authorized to join the network, the communication is encrypted and secure. Thread describes three phases a device has to go through to successfully join the network: Discovery, Commissioning and Attaching.

Although Thread initially was introduced targeting the field of home automation, in November 2016 the group expressed [42] to broaden their interest to the field of building automation as well. The OpenAIS pilot project uses Thread, but the Mbed OS Thread implementation was actively developed alongside with the Thread specification itself during this research. Hence, the proposed protocol presented in this chapter is built upon the capabilities provided by 6LoWPAN.

An overview of the Thread stack is depicted in Figure 4.2.

4.1.4 OpenAIS implementation

The OpenAIS project defines a reference architecture for Low Power Radio Access Points (LPR APs) [3]. An LPR AP enables the easy integration of the OpenAIS network into the existing IT infrastructure and it serves as a unified interface between this infrastructure and the used low power wireless PHY/MAC network technology.

![OpenAIS LPR AP protocol stack](image)

Figure 4.3: OpenAIS LPR AP protocol stack

The LPR AP architecture consists of two embedded boards: a wireless mesh Border Router and a Linux Router. This two-board design facilitates the envisioned PHY/MAC independence of the OpenAIS architecture.

The Border Router is responsible for routing between the wireless mesh network and the rest of the network on a different physical medium. The OpenAIS architecture selects IEEE 802.15.4 with 6LoWPAN and Thread as its preferred PHY/MAC constrained network, thus the border router takes the role of a 6LoWPAN Border Router as described in section 4.1.2. The Border Router supports CoAP, UDP, IPv6, 6LoWPAN and Thread.

The Linux Router is an embedded computer capable of running Linux or a Linux-like operating system providing IT management features such as: VLAN support, authentication and authorization, firewall, remote control, network management, remote logging. Although OpenAIS is completely based on IPv6, legacy systems on IPv4 are supported, hence the Linux Router also provides IPv4/IPv6 tunneling. The Linux Router supports HTTP, CoAP, TCP/UDP, IPv4, and IPv6. A possible protocol stack – based on the information in [3] – for the LPR AP is depicted in

28 Time synchronization in IoT lighting control
4.1.5 Medium access issues in the mesh

To communicate over a shared medium in a multiple access network, a media access control mechanism needs to be used. An example to this is the now obsolete shared Ethernet network variant with hubs and repeaters, where the CSMA/CD medium access method is used. CSMA/CD today is only used for compatibility with legacy equipment. Modern Ethernet networks use full duplex communication, where the switches provide isolated segments (collision-free domains), making using a MAC protocol unnecessary. This also means that the access time delay component (as defined in section 2.2.1) can be considered insignificant. The most significant delay component in an NTP message over a modern Ethernet network, hence will be the propagation time: the time the message spends waiting in message queues of the network equipment.

In 802.15.4 wireless communication the medium is shared between the nodes. The basic idea of CSMA/CA method used in these networks is the following:

1. Delay for some random interval (back-off time).
2. If the channel is idle, start sending data frame. Otherwise go to step 1.
3. Receiver acknowledges the correct reception of the data frame.
4. If the sender does not receive an acknowledgement, retry from step 1.

The number of retries at step 2 and 4 has an upper bound, when it is reached the packet is dropped. If we consider this over multiple hops, the access time and its uncertainty accumulates. Hence the assumption of NTP – a symmetrical network delay – in such networks fails. In the following section a time synchronization method is proposed for such multi-hop mesh networks.

4.2 Mesh Time Synchronization Protocol: the design

The protocol described in the following is based on the 6LNTP protocol by Hong et al. introduced in section 2.2.6. In the first phase of the protocol, the 6LoWPAN gateway corrects its time using the NTP protocol. However, there could be a scenario where a node in the mesh is connected to a more accurate time source than the 6LoWPAN gateway, for example a dual-stack node that also has wired connection, or a node equipped with a GPS stratum 0 reference clock. To include such
CHAPTER 4. MESH TIME SYNCHRONIZATION PROTOCOL

scenarios, the time synchronization role of the 6LoWPAN gateway will be referred to as Mesh Time Server for the rest of this paper.

The LPR AP present in OpenAIS (section 4.1.4) can host the role of the Mesh Time Server, as it creates and maintains the mesh. Hence, the resource-rich Linux Router in the LPR AP can use the full NTP protocol to achieve a synchronization accuracy of tens of milliseconds over the Internet, or in the order of 100 microseconds in the local network [19].

In the second, LoWPAN synchronization phase the nodes on the mesh are synchronized. As Hongs’ description lacks the implementation and design details, a proposed design is explained in this section for the second-phase synchronization.

4.2.1 Message format

All the messages are sent as UDP datagrams [28]. The message size is 10 bytes including a message sequence number (or ID), a message type, and a timestamp field. The message format is depicted in Figure 4.5.

```
<table>
<thead>
<tr>
<th>LSB</th>
<th>MSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Type</td>
</tr>
<tr>
<td>8 bit</td>
<td>8 bit</td>
</tr>
</tbody>
</table>
```

Figure 4.5: Proposed message format for the Mesh Time Synchronization Protocol

The message ID field is 1 byte. The message ID is an integer number between 0 and 255, generated by the Mesh Time Server. The message type field is 1-byte long. The 5 most significant bits are unused. For the 3 least significant bits the following combinations are defined (with LSB bit on the right):

- 001, if the message is a `REQ` or request,
- 010, if the message is a `SYNC` or synchronization,
- 011, if the message is a `CORR` or correction,
- 110, if the message is an `R_SYNC` or synchronization with reset, and
- 111, if the message is an `R_CORR` or correction with reset type of message.

The timestamp field is 8-byte long and it is in the standard NTP timestamp format as explained in section 3.1.2.

Messages with type `SYNC`, `R_SYNC`, `CORR` and `R_CORR` are always link-local multicast messages. Multicast on the link-local scope in 6LoWPAN is essentially a link-layer broadcast and is not retransmitted. The `REQ` message is always a unicast to the Mesh Time Server.

4.2.2 Time dissemination in the mesh

When a node wants to synchronize its time, it sends a unicast `REQ` message to the Mesh Time Server. The time request message may be logged by the server to adjust the synchronization schedule, but it does not have to be immediately followed by synchronization messages. The schedule of these messages may be chosen by the Mesh Time Server. This schedule is not detailed in the protocol design here, and not a topic of the present work.

In the synchronization phase the Mesh Time Server transmits a `SYNC` message to its 1-hop neighborhood. As radio waves travel at a speed of light the propagation time is considered zero. The server captures timestamp \( t_0 \), when the message is successfully transmitted via its radio, and prepares a `CORR` message with this timestamp \( t_0 \) in it. The `CORR` message is then also transmitted in the 1-hop neighborhood.
CHAPTER 4. MESH TIME SYNCHRONIZATION PROTOCOL

After receiving a SYNC-CORR message pair, a node can calculate its $\theta$ offset w.r.t. sender as the following:

$$\theta = t_0 - t_1$$

where $t_1$ is a timestamp captured with the receiver node’s local clock at the time of receiving the SYNC message, and $t_0$ is the timestamp from the received CORR message containing the sender node’s local time at the time of sending the SYNC message.

Figure 4.6 shows an example, where the SYNC message is sent and received at $t_0$ and $t_1$. The exact value of $t_0$ timestamp only known after the transmission due to medium access delays. This $t_0$ is therefore sent in the CORR message. The $\theta$ offset of the receiver w.r.t. to the sender can be calculated as $\theta = t_0 - t_1 = 135ms - 231ms = -96ms$.

![Figure 4.6: Algorithm for synchronizing time over the mesh.](image)

To disseminate the correct time in the mesh, the receiving nodes have to forward the SYNC and the corrected CORR messages. When the 1-hop nodes do this, the 2-hop nodes may receive many SYNC-CORR message pairs. Because of the nature of broadcast messages it may also happen that either the SYNC or the CORR message is not received. Hence, when a node receives a SYNC message it timestamps ($t_1$), and it stores the message, the timestamp and the sender node’s address. The node stops listening for CORR messages, when it receives one from a node, whose respective SYNC message was stored before. With a correct SYNC-CORR message pair, the node sets its time and retransmits the SYNC message in its own 1-hop neighborhood without modifications. After the message is forwarded, the node corrects timestamp $t_0$ with the time it took for the node to receive, process and send out the SYNC message and it retransmits the CORR message with the corrected timestamp in it.

As every node forwards the original SYNC and their own corrected CORR messages, the correct time eventually disseminates in the whole mesh, as depicted in Figure 4.7.

4.2.3 Time request message

A REQ or time request type message should include the current time of the sender node, so the Mesh Time Server may optimize the synchronization schedule. The ID field is unused.

4.2.4 Synchronization message

A SYNC or synchronization type message includes a timestamp with the sender’s local time at message construction.

The ID field ensures that nodes are always synchronized to the most recent messages and that old messages are not forwarded or get stuck in the mesh. The ID field is a 1-byte Serial Number field with Serial Number Arithmetic, as defined in [43]. Serial Number Arithmetic solves the problem of the wrapping of binary represented sequence numbers, but leaves some comparisons...
undefined. Therefore, to compare and decide which 1-byte sequence number is more recent 2's complement binary arithmetic should be applied, because its operations are fully defined on the entire range.

When receiving a message, a timestamp must be captured with the node’s local clock. The message should not be processed and forwarded if:

(a) it is already saved from the same sender,

(b) or it is older than the message the node last synchronized to. For this the message ID of the last successful synchronization shall be maintained internally on every node as CID (current ID).

The message is then saved together with the timestamp and the address of the sender. Saving the messages makes the protocol more robust, however, in a densely populated mesh the number of saved messages can grow unnecessarily high. A limit on this number may be selected considering the required robustness, the available memory and processing capacity, or the planned network topology. If messages with a more recent ID are received, the old messages should be deleted from the memory.

The sequence diagram of processing SYN" messages is depicted in Figure 4.8.

4.2.5 Correction message

A CORR or correction type message includes the $t_0$ timestamp, that was captured on the sender when the SYN" message was transmitted. The ID field of the CORR message contains the same sequence number as its corresponding SYN" message. When receiving a CORR message, the message should not be processed and forwarded if:

(a) it is older than the message the node last synchronized to (CID),

(b) or the sender of the CORR message is not the same as any of the stored SYN" messages' sender. As SYN" messages with the same ID are forwarded at different times, it is important that the receiver node pairs the same transmitter node's SYN" and CORR messages.
CHAPTER 4. MESH TIME SYNCHRONIZATION PROTOCOL

Figure 4.8: Processing of incoming SYNC messages

After receiving the message, the node’s offset w.r.t. the sender node is calculated as explained in section 4.2.2 and the new time is set.

When the node’s time is set, the messages can be forwarded. Before the message is re-broadcasted, a random time delay should be inserted – both before the SYNC and the CORR messages – to avoid collisions with retransmissions from neighboring nodes. When retransmitting the unchanged SYNC message, a new $t_0$ timestamp should be created, and sent in the CORR message. The sequence diagram of processing CORR messages is depicted in Figure 4.9, where the aforementioned random delay appears as a step in the handler process. It should be noted, however, that this delay should not block the processing of new messages. The motivation is to always forward messages only with the most recent ID, hence, if a SYNC-CORR message pair is received with a more recent ID, the “scheduled” forwarding of any message with an older ID should be cancelled.

Picking the minimum and maximum random time delay is quite challenging. If the first-hop nodes’ minimum random delay is zero, the maximum random delay should be big enough to give those nodes a chance to transmit. This way, the next-hop nodes’ minimum random delay should ideally be bigger than the previous-hop nodes’ maximum delay. Even if the nodes’ radio reception map is known, it is still cumbersome to calculate the minimum-maximum values for every node in the network. Such radio map is, however, not available because of the nature of wireless communication: the environment is ever-changing, nodes might move. This topic would require further research and not part of the current work. Hence, the minimum random delay was set to 0, the maximum random delay was set to 1000 ms for every node in the network.

4.2.6 Mesh Time Server

The node running the Mesh Time Server application should be accurately synchronized with the other parts of the network using NTP.

Error handling

When a node on the mesh restarts, it is in an unsynchronized state, thus it will synchronize its time with the next pair of time messages. The Mesh Time Server, however, is responsible to generate the sequence number found in the ID field of the SYNC/CORR messages. Although the sequence numbers make sure that old messages are not processed and forwarded in the network, in case of an unfortunate power loss or error on the Mesh Time Server the internal sequence counter may be lost. To handle this scenario, the $R_{SYNC}$ and $R_{CORR}$ type messages are defined. An $R_{SYNC}$ message must be processed and forwarded the same way as SYNC messages, but without condition (b), where the message ID is checked. An $R_{CORR}$ message must be processed...
CHAPTER 4. MESH TIME SYNCHRONIZATION PROTOCOL

and forwarded the same way as CORR messages, but without condition (a), where the message ID is checked. An R_SYNC message must always be followed by an R_CORR message with the same sequence number. In case of a server restart the first 5 sequence should be sent with an R-type message. Figure 4.10 shows a combined case, where both the Mesh Server and one of the nodes was restarted.

**Synchronization schedule**

In case of the SNTP implementation, the synchronization round is initiated by the client, therefore the schedule is left open to be decided by every node’s application. In case of the Mesh Time Synchronization Protocol, the rounds are initiated by the Mesh Time Server. The periodicity of these rounds has an upper bound defined by the required accuracy and the drift rate together. A tolerant minimum bound can be defined, if a new round can only happen after the previous one reached every node in the mesh: if the topology is known, the time needed to disseminate time in the whole mesh is the sum of the maximum delays for forwarding both SYNC and CORR messages multiplied by the number of hops. More strict minimums can be defined by calculating with minimum delays. The possible scheduling mechanisms are not part of the current work.
4.3 Implementation

The implementation of the Mesh Time Synchronization port number is based on the SNTP implementation discussed in section 3.2.3. The protocol requires a separate protocol and a link-local multicast address to operate. In this implementation port 4705 is selected as at the time of the research it is an unassigned port in the IANA registered ports. The \texttt{ff02::185} multicast address is selected for similar reasons. In the Mesh Time Server implementation, the request messages are simply ignored, the synchronization rounds are initiated by the application.
Figure 4.11: UML class diagram for the Mesh Time Synchronization Protocol implementation
Chapter 5

Evaluation

In this chapter the two selected protocols, the SNTP and the Mesh Time Synchronization Protocol are evaluated.

The implementation of the protocols is quite straight-forward, but since they are implemented on a hardware-independent abstraction layer, it is hard to estimate how they perform in a real scenario. As explained in section 2.2.1, the place of timestamping is crucial: every additional layer between the network interface and the synchronization application increments the error offset. However for the evaluation of the chosen time synchronization protocols, comparing their best achieved accuracy is also not enough. The best protocol has to have low impact on the communication channel, it has to align well with the requirements imposed by the architecture, or it has to offer a good compromise between these. Such a compromise was taken into account for instance, when a time synchronization protocol was selected for a specific physical medium and topology, the wireless mesh.

The architectural choices that may have an influence on the protocol’s implementability and the achievable synchronization accuracy also have to be considered. Such choice in OpenAIS is the independence of the physical layer, the support for both wired and wireless networks, the support for wireless multihop communication.

The evaluation was done on the same platform used in the OpenAIS pilot project. In section 5.1 the development platform and the measurement setup are introduced. First, the development board’s clock drift is measured. Then the accuracy is measured in best case and at various network loads. The protocols are also tested in an event-driven and in a thread-driven programming model. In section 5.2 the measurement results are presented and evaluated. It is difficult to calculate scalability without knowing the spatial topology and the applied object-model on that topology: for instance, for the synchronized control of luminaires (Feature 7 in section 1.1.3), an agreement of time between control objects is more important, than between the luminaires. However, in section 5.3 an effort was made to compare the two synchronization methods in the number of exchanged messages. In sections 5.4 and 5.5 the protocols’ security considerations and their requirements towards the architecture are discussed.

5.1 Measurement design and setup

This section introduces the FRDM-K64F development board used together with the 802.15.4 transceiver in the OpenAIS pilot project. Then the following measurements setups are described in more detail:

M1 Clock drift w.r.t. UTC reference
M2 SNTP accuracy over Ethernet
M3 SNTP accuracy over 6LoWPAN
CHAPTER 5. EVALUATION

M4 Mesh Time Synchronization Protocol accuracy over 6LoWPAN

M5 Accuracy in a congested network

M6 Accuracy with operating system delays

5.1.1 FRDM-K64F board

The Freescale FRDM-K64F (referred to as K64F) is used for evaluation, as it is the official ARM Mbed OS 3 [31][44] and OpenAIS development platform. The board has an Arduino R3 compatible pin layout, a 6-axis digital accelerometer and magnetometer, a tri-colored LED, 2 push-buttons, a microSD card slot, an Ethernet port and headers for use with 2.4 GHz radio modules such as 6LowPAN or Bluetooth. The board features an ARM Cortex-M4 core with a 1024 KB program flash memory and 256 KB RAM; programmable interrupt timers, low-power timers and a Real-Time Clock. A micro-B type USB cable can be used for power supply, serial communication, flash programming and debugging.

5.1.2 802.15.4 transceiver

The FireFly 6LoWPAN shield is used for the wireless measurements. The shield has an Arduino R3 pin layout, so it can be easily connected to the K64F board. The shield has an AT86RF233 Atmel Low Power 2.4 GHz transceiver that can be used – among others – for 802.15.4 and 6LoWPAN. It provides an SPI radio transceiver interface between the antenna and the microcontroller. While all RF-critical components are integrated in the chip, the shield provides an on-board chip antenna and a UFL antenna-connector. The transceiver chip has an internal 128 byte RAM to buffer the transmitted and received data.

5.1.3 Setup for clock drift measurement (M1)

In order to measure the drift of a crystal oscillator, its crystal’s frequency error need to be measured accurately. Although the literature describes several ways to do this, the motivation was not to measure the board’s crystal error, but rather to get an idea about the implemented clock application’s drift from its origin UTC reference over time. Hence, the time kept on the client was compared to the time on the stratum-1 NTP server.

![Figure 5.1: Clock drift measurement setup (M1)](image_url)

In the measurement setup [M1] depicted in Figure 5.1, the K64F board synchronizes its time using the Mbed 5 SNTP client implementation. The NTP server runs on a Raspberry Pi computer.
with a version 3 Adafruit Ultimate GPS module directly connected to it. The GPS module provides a \textit{stratum-0} UTC time reference, but it sends the time information using serial communication through UART to the Raspberry Pi. This does not provide an accurate timing signal. Hence, the used module also includes a separate pulse-per-second signal, which marks the start of the second. The used module’s pulse-per-second signal has a 10 nanosecond jitter.

In the M1 measurements, the K64F board first synchronizes its time to the NTP server over a network router. The router functions as a switch and it is also responsible to send out Router Advertisement messages and configure the hosts with stateless DHCPv6. After a successful synchronization, the Raspberry Pi captures a timestamp every 10 seconds. When the timestamp is captured one of the Raspberry Pi’s output port is set to high signal level for a short period of time, generating a rising edge. This was done by an application written in C-code by reading the system time and using the WiringPi \cite{wiringpi} library to access the output port. Benchmarks were performed in \cite{benchmarks} to measure the delay on the Raspberry Pi between sending a command to the output port and the actual change of value on the output port. For the WiringPi module this value was found to be always sub-1-microsecond, therefore this delay may be ignored. The generated signal is connected with a short wire to an interrupt input port of the K64F board. The K64F timestamps whenever the respective interrupt routine is invoked. A source of error here might be how quickly this interrupt is processed (eg. interrupted by higher priority interrupts), but this was considered to be not significant (in the order of sub-1-microseconds) in the measurements. The timestamps are then collected from the K64F board via serial line, and from the Raspberry Pi via network SSH. The measurements were conducted at room temperature.

### 5.1.4 Setup for SNTP accuracy over Ethernet measurement (M2)

In the M2 setup, the SNTP client, implemented on a K64F board, synchronizes its time using the SNTP client implementation. The NTP server is connected to the internet and synchronizes its time to several upstream servers. The K64F board and the NTP server are connected to the same network through a router, similarly as explained in section \ref{5.1.3}. After a successful synchronization, both the server and the client captures a timestamp, as explained in section \ref{5.1.3}. The timestamping has to happen shortly after a successful synchronization, to exclude error from the drift. The measurement setup is depicted in Figure \ref{fig:snntp_setup}.

![SNTP synchronization accuracy measurement setup for Ethernet connection (M2)](image)

**Figure 5.2:** SNTP synchronization accuracy measurement setup for Ethernet connection (M2)

It is important to note that after the K64F is restarted, the first-time synchronization is not
included in the measurements. To send a message to a specific address after the socket is opened, the network stack first need to send out IPv6 Neighbour Discovery messages. These can introduce a significant asymmetric round-trip delay, but since they are cached, they will only happen at the first request (or when not present in the cache). Similarly, if DNS name resolving is used, the IP-address should be acquired before passing the message on to the network stack. It is also important to note that debug messages (traces) on the serial line should be completely disabled during the message exchange as they create a significant delay. The ARM Mbed platform includes such debug messages for socket and network events.

5.1.5 Setup for SNTP accuracy over 6LoWPAN measurement (M3)

Figure 5.3: SNTP synchronization accuracy measurement setup for 6LoWPAN multi-hop connection (M3).

In M3, the SNTP client implementation is tested over a wireless multi-hop communication. As explained in section 4.1.5, in case of a shared-channel wireless communication, a medium access protocol needs to be used. This may introduce a significant access time. The motivation was to measure how this affects the accuracy in practice over one (star topology) or more hops (mesh topology). The measurement setup is similar to the previous ones, but the K64F board, which is connected over Ethernet to the same network as the NTP server, runs a 6LoWPAN Border Router [47] application. Further K64 boards are connected to the network over the Border Router wirelessly. Nanostack, the 6LoWPAN implementation used in the measurements, is using the RPL routing protocol. RPL enables the network to disseminate information over dynamic routing topologies in a constrained environment. The routing should be kept static and controlled over several measurements, but the closed-source nature of the used network stack implementation limited this to a few scenarios. From the possible scenarios the following two were used:

**Scenario 1** Lower the transmit power of the radios and place the nodes spatially distant from each other, creating a line topology.

**Scenario 2** Filter out packets on the link-layer based on their MAC address using the available APIs.

In case of Scenario 1, the nodes’ radio power were lowered to the minimum 20µW (corresponds to −17dBm), and placed with 3m distance between each hop in an office. This scenario turned out to be very unstable: the nodes frequently changed routing between neighbouring nodes due to overhearing, and the number of retransmissions was very high due to interference from other 2.4GHz transmitters in the office (WiFi, and Bluetooth).

In case of Scenario 2, each node was set up to drop any packets that are sent by their non-direct
neighbors. This setup was possible due to a “MAC filter” API available with the network stack. The accuracy was measured at each hop similarly as in M2. In this setup the nodes were all placed on the same table a few cm distance from each other. The radio transmitter was set to the default 1mW (0dBm). This measurement setup is depicted in Figure 5.3.

5.1.6 Setup for Mesh Time Synchronization Protocol accuracy over 6LoWPAN measurement (M4)

In M4 all the nodes on the mesh are running the Mesh Time Synchronization Protocol. They do not synchronize to an external server, but to the Mesh Time Server. In the measurement setup, the Mesh Time Server was placed on the Border Router, as this is the recommended placement for OpenAIS too: the LPR AP may have the full NTP implementation running on it, thus it can synchronize its time with less than a millisecond accuracy to the rest of the local network (as mentioned in section 2.2.4). The measurement setup was done similarly as in M2 and M3 directly after each synchronization round, but this time the 6LoWPAN Border Router was generating the rising edge connected to the mesh nodes’ interrupt pin. The setup is depicted in Figure 5.4.

Figure 5.4: Mesh Time Synchronization Protocol accuracy measurement setup for 6LoWPAN multi-hop connection (M4)

5.1.7 Setup for accuracy measurement in congested network (M5)

Two computer’s clock can be synchronized, if one of them is selected as a reference, and this reference tells its current time to the other one. When this is done over a computer network, the message with the current time may be delayed in the network equipment until it reaches the other computer. As seen in Chapter 2, time synchronization protocols tackle the problem in various ways. NTP, for example, assumes an equal delay both ways between server and client to calculate the offset, thus any difference from the symmetrical delay will reduce the accuracy with the amount of half of the difference between request and response message travel times. When the network load is high, the message latency grows as messages need to spend more time in the network buffers and queues. This may delay every message differently, hence when using a time synchronization protocol, this has to be taken into consideration.

In M5 the motivation was to simulate the effects of a congested network, with a focus on delaying messages by keeping them queued. In the SNTP measurements, the messages were delayed directly on the NTP server: the queuing discipline mechanism (qdisc) available in the Linux kernel traffic control (tc) subsystem is capable of delaying the outgoing and incoming messages in a different manner. The qdisc module can be added and removed in run-time, making it an easy and flexible solution in the measurements. In the measurements of the Mesh Synchronization Protocol, the messages were delayed in the synchronization application on every node by modifying the minimum random delay value before forwarding a message.
5.1.8 Setup for operating system delays measurement (M6)

The goal of M6 was to implement time synchronization on the application layer of Mbed OS, without using any microcontroller-dependent feature. Mbed OS abstracts the hardware-dependent implementations by providing an object-oriented interface to the application developer. This enables a modular application architecture and transferring code when switching to a new – but supported – microcontroller.

Mbed OS 3 and 5, however, use two different programing models: while Mbed OS 3 uses an event-driven, Mbed OS 5 uses a multi-threaded model. This might have consequences on timestamping in the application layer. In the event-driven model, operations (such as sending the network message) are non-blocking, since events do not have a priority, their results are appended to the end of an event queue. If the time an event spends in the event-queue is too high, timestamping will be delayed. It is almost impossible to predict when and how long this queue can be. The motivation of M5 was to measure how the event-queue affects time synchronization.

In M5, a periodic task was scheduled with the MINAR scheduler to be called every 10 milliseconds. The task’s execution time was changed between 1 ms, 5 ms and 9 ms to simulate a small, a medium and a heavy load on the event-queue.

In the multi-threaded model, the developer may decide freely if an operation is blocking or non-blocking. If the synchronization is done in a blocking way, other threads cannot affect the synchronization. Unless a higher priority task is started during the synchronization task. In this and in case of non-blocking mode, when other threads are allowed to run while the synchronization thread is waiting for a server response, the timestamping can be still correctly done in an interrupt routine.

5.2 Results

In this section, the results for M1, M2, M3, M4, M5 are presented and evaluated. In the first subsection the results for the clock drift measurements (M1) are presented. In the second subsection the best-case results for SNTP are presented both over Ethernet (M2) and 6LoWPAN (M3). Following this, the results with a congested network (M5) and on different platforms (M6) are presented. In the third subsection the results for the Mesh Time Synchronization protocol are presented in a similar manner. These measurements could be conducted over 6LoWPAN only (M4, M5 and then M6).

5.2.1 Clock drift

![Image](image-url)

Figure 5.5: Measurement results for clock drift

The results of the M1 clock drift measurement are depicted in Figure 5.5 and they are compar-
able to the crystal characteristics depicted on Figure 2.3. This crystal is used on the K64 board as well. The figure shows, that if the required accuracy is e.g. 40 ms, then the synchronization has to be repeated at least every 30 minutes. This, however, assumes a constant temperature in that period, which is not the case at luminaires. Applying a software or hardware-based temperature-compensation when the temperature changes so drastically, is highly recommended.

5.2.2 SNTP

All the measurements were repeated 100 times. The results are represented on box-and-whisker diagrams. The box is drawn around the 25th and 75th percentiles of a data set. The median of the data set is represented by the bar inside the box. While the lower whisker shows the minimum value, the upper whisker shows the maximum value among the results. Outliers were not removed from the data, unless otherwise is stated, as they had little impact on the diagrams.

Accuracy

Figure 5.6 shows the results for the accuracy measurements. All measurements were conducted on the Mbed 5 implementation, the wired measurements were taken in setup M2, the wireless measurements in setup M3 with the MAC-filtering Scenario 2.

The results over Ethernet show a 58µs median with a total range of 30µs. As expected, the wireless results show a growing uncertainty over a growing hop count: both the total range between the whiskers and the interquartile range represented by the box roughly doubles with every hop. The distribution is symmetric in every case, since the results are evenly split around the median. The results show the most ideal case: there’s no other communication going on on the network, the interference is low, messages need to be retransmitted rarely. For comparison the wireless measurements in M3 were repeated with Scenario 1, where the transmit power was lowered to minimum. The results are depicted in Figure 5.7.

It is hard to see any consistency in the data: results spread over a wide range between −20 and 25ms for the first three hops, while the 50% of the data ranges between −5 and +5ms. The fourth hop shows a big jump in ranges. With the radio power lowered to the minimum, the effects of path loss are inevitable: reflections, diffraction, interference with other signals (Wi-fi, Bluetooth). During the measurements the nodes often lost signal and disconnected from the network. The results seem random, so the main outcome from this measurement is that SNTP becomes almost unpredictable in case of bad wireless link quality. Further wireless measurements were all taken with the MAC-filtering Scenario 2 using the default transmit power (1mW).
CHAPTER 5. EVALUATION

Network congestion

The results for M5 in a growing network delay over Ethernet are depicted in Figure 5.8, with the boxes strongly squeezed due to the small range of values. The chosen network delays were 1ms, 10ms, 50ms and 100ms. The results show no surprise: while NTP works correctly if the two-way network delays are equal, the half of the difference between the two delays will appear as a synchronization error. Hence, for 1ms delay, the median accuracy is 562µs, for 10ms delay it is 5.06ms, for 50ms delay it is 25.06ms, and for 100ms delay it is 50.06ms.

These imply, that network congestion should be avoided in order to reach an accurate synchronization. If the lighting network is on a separate state-of-the-art Ethernet network, without special network-hungry (such as distributed database and computing) applications running on it, this shouldn’t pose extra challenges towards the architecture.

The results can be transposed to the wireless setup. The implications are more challenging there: due to the shared medium among the nodes, the medium access back-off times, retransmission and forwarding delays can add up easily to the tested delay values even in normal circumstances. The achievable accuracy will depend highly on the number of nodes, their topology and the number of messages going through the network, but in any case, it will never be optimal.
Platform

The results for M1 conducted on Mbed OS 3 and Mbed OS 5 are depicted in Figure 5.9. The median values are 7\( \mu \)s on Mbed 3 and 58\( \mu \)s on Mbed 5. The results might show the multi-threaded Mbed 5 as the loser in this comparison, but the reasons behind it are quite prosaic: the Mbed 3 socket interface provides an event-callback, when a message is available in the buffer, and also when a sent message has left the interface. The first one allows to timestamp before reading the message from the buffer (NTP Receive Timestamp). The second one allows to create a corrected timestamp when the request is actually sent (NTP Transmit Timestamp). The same features are currently not available in Mbed 5, but they may be implemented in a future, or a forked version of the socket interface in the form of interrupts. In the measured implementation on Mbed 5, timestamping was done after the 40-byte packet was read from the buffer, hence the 65\( \mu \)s difference.

![Figure 5.9: SNTP synchronization accuracy over Ethernet on different Mbed platforms](image)

If the socket implementation on a multi-threaded platform provides interrupts when a message is received or sent, the timestamping can be done very accurately. On an event-driven platform these interrupts appear in the form of events, that have to wait in the even-queue until they are processed, thus introducing unwanted delays in timestamping. To test a busy event-queue, a delay subroutine was scheduled to be called every 10\( \text{ms} \), but its runtime was changed to 1\( \text{ms} \), 5\( \text{ms} \) and 9\( \text{ms} \) respectively between measurements.

Figure 5.10 shows a wide range of values: from −0.5 to 0.5\( \text{ms} \) at 1\( \text{ms} \), from −2.5 to 2.5\( \text{ms} \) at 5\( \text{ms} \), from −4.5 to 4.5\( \text{ms} \) at 9\( \text{ms} \) delay. By looking at the squeezed boxes together with the wide ranges, we may conclude that, in most cases the synchronization is not affected by the delay subroutine. In this case, however, the used diagram type is not suitable to visualize the results well, so in this case the outliers were calculated. An outlier expresses a result that significantly deviates from the other results, either caused by measurement error, random deviation, or because they indicate something significant in the data. To calculate outliers, Tukey’s method was used: whiskers extend to data points that are less than 1.5 * IQR away from the upper and lower quartiles, the rest is considered an outlier and marked with a circle. IQR stands for the interquartile range, calculated as the difference between the 75th and 25th percentiles.

With this method, the results are depicted in Figure 5.11. Per data sample one of below four cases occurs, and can be seen in the figure as well:

1. The delay subroutine is not invoked during the message exchange, so it has no effect on the synchronization.
2. The delay subroutine is invoked an equal times both before sending and receiving the messages, so the algorithm is able to correct it.
3. The delay subroutine is invoked before timestamping the sent request message, so it causes an asynchronous delay with a positive offset.

4. The delay subroutine is invoked before timestamping the arrival of the response message, so it causes an asynchronous delay with a negative offset.

In the event-driven model, task execution time should be kept as short as possible. Sometimes this cannot be avoided: for example the generation of encryption keys can take a considerable amount of time. In these situations a time synchronization round should not be scheduled.

5.2.3 Mesh Time Synchronization protocol

All the measurements were repeated 100 times. The 6LoWPAN Border Router was running the Mesh Time Server application and the nodes were running the node application.

Accuracy

The results acquired in M4 for the Mesh Time Synchronization Protocol on the Mbed 5 platform over 6LoWPAN are depicted in Figure 5.12. The median accuracy is 0.74ms for the first
CHAPTER 5. EVALUATION

hop, 1.60ms for the second hop, 2.43ms for the third hop and 3.28ms for the forth hop. An approximately 0.8ms error can be observed at each hop. The constant and approximately equal value shows, that the error is not due to changes in the physical link, but because of inaccurate timestamping happening due to a delay in the network stack or the Mbed platform somewhere. The reasons for this are not entirely clear and they would require further investigations on the platform. If, for instance, the error comes from a constant instruction path in the code, the error could be safely compensated with a −0.8ms offset.

Network congestion

The Mesh Time Synchronization Protocol already contains a random delay at forwarding messages by design. Tests were run with modifying the minimum random delay value to the same 1ms, 10ms, 50ms and 100ms values as at the SNTP measurements in M5, but as expected they had no effect on the accuracy.

Platform

The M1 and M2 accuracy measurements were repeated with the Mbed 3 platform. The results are depicted side-to-side with the previous Mbed 5 results on Figure 5.13. The median results are 0.48ms, 1.01ms, 1.50ms, 2.06ms over 1, 2, 3 and 4 hops respectively. Interestingly, Mbed 3 comes out as the winner again, but it has similarly a constant accumulating offset over the hops. As Mbed 3 and Mbed 5 uses the same networking stack implementation (Nanostack), the investigation could be reduced finding the answer in the different socket implementations.


5.3 Scalability

Scalability can be measured and defined in many dimensions. In case of a network time synchronization protocol scalability can be defined as how the load on the network changes with a growing number of devices, or with the need for a higher accuracy. Since the communication paths and delays are often changing and dependent on the topology and used underlying physical layer, it is hard to express a specific metric of interest. The question of scalability is discussed from two aspects: the wired, Ethernet network and the wireless 6LoWPAN mesh network.

NTP packets are small and take little time to process on the server machine. The traffic overhead is also not a concern on the wired Ethernet lighting network, as it is lightly loaded, the time synchronization doesn’t have to share bandwidth with demanding applications (such as multimedia-streams).

The server-client mode of the NTP/SNTP protocol requires 2 messages to synchronize per device. In the broadcast mode, each client receives one message in a regular interval, but it also has to do server-client mode synchronization (with a bigger interval) to calculate the round-trip delay.

In case of the wireless mesh network, sending messages over multiple hops does not scale well: throughput can decline by 50% per hop [49]. By taking a vague 6LoWPAN mesh network topology, the number of messages necessary to synchronize time using SNTP and the Mesh Time Synchronization Protocol can be calculated. Considering 5 nodes present at every hop with a maximum 4-hop distance, it can be depicted as below:

<table>
<thead>
<tr>
<th></th>
<th>Border Router</th>
<th>1st hop</th>
<th>2nd hop</th>
<th>3rd hop</th>
<th>4th hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNTP</td>
<td>2*5 + 30 = 40 messages</td>
<td>2*5 + 20 = 30 messages</td>
<td>2*5 + 10 = 20 messages</td>
<td>2*5 = 10 messages</td>
<td></td>
</tr>
<tr>
<td>Mesh Time Sync</td>
<td>2 messages</td>
<td>2*5 = 10 messages</td>
<td>2*5 = 10 messages</td>
<td>2*5 = 10 messages</td>
<td></td>
</tr>
</tbody>
</table>

In case of SNTP, considering an NTP server outside the mesh network with a client-server communication mode, the counting can be followed from the right to left: the 5 nodes the need 2*5
messages to synchronize their time. The 5 nodes in the three-hop distance need 2*5 messages again, and they also have to retransmit the messages from/to the 4th hop nodes. The same is true for the lower hops: the nodes there and the number of retransmissions needed from the previous hops add up because of the forwarding. This way the total number of transmissions on the link-layer is at least 100.

In case of the Mesh Time Synchronization protocol the scenario is explained from left to right: the Mesh Time Server on the Border Router sends out 2 broadcast messages. Then at every hop, every node retransmits these two messages. The total number of transmissions in this setup is 42.

It should also be noted that in case of SNTP, the calculated number is the best case, when no retransmissions are necessary between two nodes due to a lost packet. In case of the Mesh Time Synchronization protocol it is both the best and worst case, as broadcast messages are never retransmitted.

If we take the results for clock drift from section 5.1.3 together with these numbers, we can estimate the number of necessary messages over a period of time to maintain synchronization within the given requirements. Hence looking at Figure 5.14, we see that if the required accuracy is 15\(\text{ms}\), SNTP would require around 500 messages per hour, while the Mesh Time Synchronization protocol would require around 200 messages per hour with the previously described topology.

(Note, that with both protocols perfect synchronization accuracy is assumed, hence the growing offset over hops for the Mesh Time Synchronization Protocol are not considered.)

![Figure 5.14: The number of required messages to be transmitted per hour in the example mesh depending on the required accuracy](image)

5.4 Security considerations

The data available in the NTP or Mesh Time Synchronization packets can be considered public, so there’s no need to disguise or encrypt the data. By listing possible threats, one goal of an attacker may be to produce incorrect or inconsistent time values. This would disrupt or cause failure in time-dependent applications: in case of an OpenAIS system, considering its features mentioned in section 1.1.3, this may mean corrupted logs (Feature 1), incorrectly timestamped sensor events (Feature 2), or a disrupted user experience (Feature 3, Feature 5, Feature 6, Feature 7). Another goal could be to overload and then break down the network with synchronization messages.

SNTP does not authenticate the server by default, but it can be enabled through the extension fields in the packet. The 4th version of the protocol supports both symmetric and public-key cryptography [50]. The comparison of the two for constrained nodes is out of the scope of this report. The attack vectors in case of SNTP synchronization may be the external reference NTP servers, the local NTP server(s), the local LAN network, or one of the hosts running the SNTP client application. The local server always synchronizes its time to several upstream serves. If a server is compromised, it is automatically marked as an outlier w.r.t. the other servers, hence the
majority of the reference time server has to be compromised for this attack. The local NTP server may get compromised. A possible countermeasure is to randomize the server hosts synchronize to before every new synchronization round. If the local LAN network or a client host is compromised, the attacker may delay or stop synchronization messages or break down the network. Slightly delayed synchronization messages may remain undetected if the attack takes place over a longer period of time, modifying the clock in small steps.

Mesh Time Synchronization Protocol messages are present only on the wireless network. OpenAIS requires all wireless messages to be encrypted on the link-layer. The attack vector is common with the SNTP attack vector up to the Border Router, where the Mesh Time Server is located. The vector then includes the Border Router, or a node on the network. If the Border Router is compromised, the whole mesh is compromised. In case of a compromised node on the network, that node might attempt to send out \textit{SYNC-CORR} message pairs. Although the messages are forwarded to all the nodes, the breach may remain undetected if the nodes are not placed in a dense topology, thus leaving single-point-of-failure situations open.

5.5 Requirements towards the OpenAIS architecture

For the SNTP synchronization, a fully-implemented, local NTP server is recommended. The NTP server should be strategically placed, so the network delay is as low as possible for every node. The same physical machine as where the LWM2M Server (software) is running on in the building’s intranet may be considered for this function.

For automatic service discovery the multicast and server pool schemes are recommended. The multicast scheme uses the native IPv6 multicast, where the NTP server regularly sends multicast messages to the IANA assigned \texttt{ff05::101} site-local multicast address. The forwarding of this address has to be configured in the network equipment. In the server pool scheme, the client is configured with a (locally resolvable) DNS address. This scheme provides flexibility, when the server is moved to another host, or when multiple servers are configured. In the latter case, the returned IP address can be randomized or optimized based on the location of the client. For this scheme, a local DNS server has to be configured.

It should be noted, that synchronizing SNTP devices with an external NTP server (over the internet) is discouraged due to the lack of complex filtering and adjustment algorithms included in the full NTP protocol. Configuring hundreds/thousands of clients to a DNS address or a direct IP address owned by another entity is considered as abusive behavior.

For the Mesh Time Synchronization Protocol, the Mesh Time Server should be placed strategically to a place where high-accuracy time is available, such as the LPR AP. The LPR AP includes a resource-rich Linux Router and a Border Router. The nodes on the network always know the Border Router’s address, so they should be able to send out time request messages without problems. Nodes without support for the Mesh Time Synchronization Protocol may be present on the network, if every node using the protocol have at least one neighbor supporting the Mesh Time Synchronization Protocol. Single-point-of-failure situations should be avoided, by placing the nodes in a dense topology.
Chapter 6

Conclusions

As Cisco [1] mentions, IoT is at a stage where disparate networks and sensors must come together and interoperate under a common set of standards. Standards must be defined in the areas of architecture, communications, security and privacy especially. OpenAIS pursues the standardization of a lighting system architecture.

Keeping a synchronized, global notion of time across these newly connected networks represents a new area of research. One of the goals of this research was to select the protocol, that may achieve a milliseconds accuracy without the close coupling of software and hardware, thus taking advantage of open software standards over proprietary solutions, on top of IPv6.

Mills’ Network Time Protocol is the most common protocol to synchronize hosts over the internet, thus it is advantageous to investigate its applicability in this new context. The advancement of technology and new protocols enabled even the tiniest devices to connect to the internet, but protocols designed for resourceful hosts might perform poorly on them. Therefore besides NTP, the Mesh Time Synchronization Protocol is proposed to extend NTP’s capabilities to IoT-typical constrained links.

6.1 Contribution

As mentioned in 2.2.4, NTP should be able to synchronize time with an accuracy of tens of milliseconds over the internet and in the order of 100 microseconds in local networks. The implemented SNTP version with the full on-wire protocol achieved a median accuracy of 7µs on the event-driven Mbed OS 3 and 58µs on the multi-threaded Mbed OS 5 synchronizing to a local NTP server over Ethernet. It was shown that – as expected – the implementation performs badly in multihop 6LoWPAN networks. It was shown that SNTP only performs well if the two-way message delays are equal, thus both operating system and network delays should be avoided. This research shows, that SNTP can be used for millisecond-accurate synchronization if an NTP server is available on the local network.

The proposed Mesh Time Synchronization Protocol for wireless mesh networks can synchronize nodes on a 6LoWPAN network to the global UTC reference time with a millisecond accuracy. The results show a median accuracy of 0.48ms, 1.01ms, 1.50ms, 2.06ms on Mbed OS 3 and 0.74ms, 1.60ms, 2.43ms, 3.28ms w.r.t. the Mesh Time Server on Mbed OS 5 over 1, 2, 3 and 4 hops respectively. The synchronization accuracy of this protocol is not affected by network congestion or unpredicted operating system delays.

Neither SNTP, nor the Mesh Time Synchronization Protocol compensates between synchronization rounds for the crystal-oscillator originating drift. Therefore, the rounds have to be repeated regularly. SNTP has little impact on modern Ethernet networks. This is, however, not the case on a low-bandwidth mesh network. When SNTP is compared on an example 6LoWPAN mesh topology to the Mesh Time Synchronization Protocol, the former one needs at least 2.5 times more transmissions.
CHAPTER 6. CONCLUSIONS

Because of the OpenAIS requirements towards the wireless physical layer, Mesh Time Syn-
chronization Protocol messages are encrypted on the link-layer by default. SNTP messages over
Ethernet may be authenticated with symmetric or public-key cryptography.

The research contributes with its unique measurement setups, to compare time between an
embedded development board and an NTP reference time server running on a Raspberry Pi by
exploiting the devices’ GPIO ports and interrupt signals.

6.2 Future work

A main motivation for introducing the Mesh Time Synchronization Protocol was to show that
time can be synchronized to the global UTC reference even over constrained links, taking into
account limited bandwidth or memory as well. Although this first version of the protocol achieves
a fairly good accuracy, it also leaves some questions unanswered.

- The measurement results show a constant 0.5\(ms\) offset on Mbed 3 and 0.8\(ms\) offset on Mbed 5 accumulating over hops. The reason to this was not discovered during the research. If this offset could be proven to be caused by a constant series of instructions delaying the timestamp, the offset could be safely compensated.

- Section 4.2.2 and 4.2.3 discusses that time request messages are sent to the Mesh Time Server. The Mesh Time Server may optimize the synchronization schedule based on these requests. The schedule is not detailed in the present work, however, it has an important role, as the network throughput is limited and dissemination of the current time in the whole network happens with a delay.

- The 802.15.4 network uses the CSMA/CA mechanism to access the wireless medium. The mechanism has an upper limit on the number of retries (4 by default), messages are dropped when this upper limit is reached. If many nodes want to transmit similarly around the same time, packets might be dropped. To overcome this, the protocol includes a random time delay before forwarding synchronization and correction messages, but it does not specify the minimum and maximum values of these delays. Further research could be done about choosing these values.

- (S)NTP works with UDP, using its own message format and port number. With the same motivation – keeping the time synchronization independent from the other features – the Mesh Time Synchronization Protocol was implemented also on UDP with its own message format and port number. The CoAP application-layer protocol \cite{4} is used to communicate between devices in the OpenAIS architecture. CoAP (similarly to HTTP) allows a number of options to be included with any CoAP message. The \textit{SYNC} and \textit{CORR} messages could be piggybacked in the ongoing CoAP messages. This might be beneficial, unless this would grow the messages to be transmitted in multiple frames, thus potentially increasing the number of re-transmissions (if one frame is lost, all the frames have to be retransmitted). The Mesh Time Synchronization also requires sending 2 consequent messages – this might not happen for a very long time simply because of the applied communication model. This would force the nodes to send “empty” messages only containing the necessary option field. The extra complexity of this solution might also reduce the accuracy of timestamping.
References


54 Time synchronization in IoT lighting control

REFERENCES


[34] Mbed low-level HAL API. URL: https://github.com/ARMmbed/mbed-hal (visited on 22/03/2016) (cit. on p. 23).

REFERENCES


Appendix A

Software versions

The measurements were conducted using the following ARM Mbed modules available in the ARM Mbed repository¹:

**Mbed 3**
- mbed-drivers: v1.5.0
- mbed-hal: v1.3.0
- sockets: v1.3.0
- mbed-drivers: v1.0.0
- sal-stack-nanostack: v5.0.4
- k64f-border-router v1.0.1

**Mbed 5**
- mbed-os v5.1.0
- k64f-border-router v1.0.1

¹https://github.com/ARMmbed/