DIPLOMA THESIS

Distributed synchronized measurement system in the ARROWHEAD Internet of Things Framework

Submitted at the Faculty of Electrical Engineering and Information Technology,
Technische Universität Wien
in partial fulfillment of the requirements for the degree of
Diplom-Ingenieur (equals Master of Sciences)

under supervision of

Ao.-Prof. DI Dr. Thilo Sauter

Institute of Computer Technology (E384)
Technische Universität Wien

by

Jeronimo Govinda Mitaroff-Szécsényi
Matr.Nr. 0426219
Hetzendorfer Strasse 58-60/9/1 1120 Wien

November 4, 2017

_____

Abstract

With the development of communication in embedded systems, machine to machine communication becomes more and more important. The number of devices connected to the internet is growing exponentially, and intelligent devices are reaching more application areas than before. This leads to a fast growing complexity in all networked systems.

Demands on manufacturing are also growing. The ability to dynamically adapt a manufacturing system to current needs has become a requirement to stay competetive. However, traditional systems are designed top-down, with each component selected to fulfill its specific purpose. This makes dynamic system reconfiguration a very challenging task that is proportional to the complexity of the system. The Arrowhead Framework is a proposed solution to manage this complexity by using a service oriented architecture.

In this work a distributed synchronous measurement system within the ARROWHEAD framework [BFK+14], using the wireless 6LoWPAN MULLE platform [JVE+04] developed for this framework, is implemented. While the ARROWHEAD framework focuses on communication between devices, the main challenge remaining for a distributed measurement system is clock synchronization, especially with low-cost Internet of Things devices.

Therefore the focus of this work is to analyze the unique challenges that arise from software based clock synchronization, to develop mathematical error models and to propose methods to compensate these errors. The implementation of this work is based on the PTP protocol [IS08], however all the findings can be applied directly to all software based synchronization protocols. Software only timestamping leads to undetectable jitter from network conflicts and interference from other processes. Additionally the available timestamping points lead to transmission delay asymmetry. It will be shown that, by understanding these errors in detail, software based clock synchronization can be greatly improved.

Kurzfassung

Durch die Vernetzung von integrierten Systemen wird Maschinen zu Maschinen Kommunikation immer wichtiger. Die Anzahl an Geräten die mit dem Internet verbunden sind wächst exponentiell, und intelligente Geräte finden regelmäßig neue Anwendungsgebiete. Dies führt zu einer rasch wachsenden Komplexität in vernetzten Systemen.

Anforderungen an die Industrie wachsen ebenfalls. In einem sich rasch ändernden Markt muss ein Hersteller seine Fertigungsprozesse dynamisch anpassen können um konkurrenzfähig zu bleiben. Traditionell werden Systeme allerdings Top-Down entworfen, mit jeder Komponente die ihren speziellen Zweck erfüllt. Dies macht eine Neukonfiguration zu einer Herausforderung welche proportional zur Komplexität des Systems ist. Das ARROWHEAD Framework ist eine vorgeschlagene Lösung dieses Problem durch eine serviceorientierte Architektur zu lösen.

In dieser Arbeit wurde ein verteiltes, synchrones Messsystem, basierend auf dem ARROWHEAD Framework [BFK+14] und der dafür entwickelten 6LoWPAN MULLE Plattform [JVE+04], implementiert. Während das ARROWHEAD Framework die Kommunikation der Geräte untereinander beschreibt bleibt Uhrensynchronisation, besonders bei günstigen Internet der Dinge Geräten, eine große Herausforderung für verteilte Messsysteme.

Der Fokus dieser Arbeit ist daher die Analyse der besonderen Herausforderungen die durch softwarebasierte Uhrensynchronisation entstehen, die Entwicklung einer mathematischen Beschreibung dieser Fehler sowie Methoden zur Fehlerkompensation zu entwerfen. Die Implementierung der Uhrensynchronisation in dieser Arbeit basiert auf dem PTP-Protokoll [IS08], die Erkenntnisse können allerdings direkt auf alle softwarebasierenden Uhrensynchronizationsprotokolle übertragen werden. Durch die reine Software Implementierung entstehen nicht-erkennbare Synchronisations-Fehlerquellen, z.B. variable Verzögerungen durch Netzwerkkonflikte oder Interferenz durch andere Softwareprozesse. Weiters führt die Limitierung des Zeitstempelpunktes zu einer Asymetrie der Übertragungsverzögerungen. Es wird gezeigt, dass durch genaue Verständnis dieser Fehler softwarebasierte Uhrensynchronisation stark verbessert werden kann.

Acknowledgements

This work has been supported by the Austrian Research Promotion Agency (FFG) under grant number 853456 (FASAN: Flexible Autonome Sensorik in industriellen ANwendungen).

Table of Contents

Abbreviations

6LoWPAN	IPv6 over Low power Wireless Personal Area Network
ACK	ACKnowledgement
ADC	Analog to Digital Converter
AVL List	Anstalt für Verbrennunskraftmaschinen List
CAN	Controller Area Network
CENTOS	Community ENTerprise Operating System
COAP	COnstrained Application Protocol
CPU	Central Processing Unit
CS	Clock Synchronization
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
DAC	Digital to Analog Converter
DNS-SD	Domain Name Service - Service Discovery
ETFA	International Conference on Emerging Technologies and Factory Automation
FPU	Floating Point Unit
GND	GrouND voltage level
GPIO	General Purpose Input Output
HTTP	HyperText Transfer Protocol
HW	HardWare
I^2C	Inter-Integrated Circuit
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISO	International Organization for Standardization
ISR	Interrupt Service Routine
LAN	Local Area Network
MAC	Media Access Control
NIC	Network Interface Card
NTP	Network Time Protocol
OMA-LwM2M	Open Machine Alliance - LightWeight Machine To Machine protocol
ppm	Parts Per Million
PTP	Precision Time Protocol
RAM	Random Access Memory
RBS	Reference Broadcast Synchronization
RTC	Real Time Clock
SLIP	Serial Line IP
SOA	Service Oriented Architecture
SPI	Serial Peripheral Interface

SW	SoftWare
TCP	Transmission Control Protocol
TSPN	Timing-Sync Protocol for sensor Networks
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
uIP	micro Internet Protocol
URI	Uniform Ressource Identifier
USB	Universal Serial Bus
USB-OTG	Universal Serial Bus - On the Go
V_{cc}	Voltage at the Common Collector
vs	versus

1 Introduction

Embedded systems are becoming smaller, cheaper, more efficient and are used more and more. A major development has been the fact that these devices are now incorporating wireless communication solutions. Removing the need for cabling allows for cheap autonomous and flexible distributed systems, and global connections allow for almost limitless applications. This has resulted in the exponential growth of the Internet of Things.

This development has led to a fast growing complexity in networked systems. Rather than designing a system once and using it for decades modern manufacturing systems require flexibility, the ability to grow or change in accordance with their demands. With these requirements traditional top-down design approaches do not work, as any major changes would require a complete redesign of the system. Instead flexible architectures are used to manage the system, using machine to machine communication to dynamically reconfigure the system.

One of these architectures is the Service Oriented Architecture, which flips the complexity problem. Instead of being configured for a specific purpose, each device offers its functions as so-called services. In its basic state it is not important what it is used for, and where it will send its data. Combined with a framework for discovery, configuration and security this approach allows for very dynamic systems to assemble.

But not only communication needs to be managed. Distributed measurement or control networks require a global notion of time. Clock synchronization is a major core service for such systems. But synchronization can be influenced by the whole system, and providing high accuracy while still keeping costs down can be a major challenge.

1.1 Problem statement

In a cooperation between the Technical University of Vienna and AVL List, an automotive engineering firm, a distributed measurement system has to be developed. The goal is to make the automotive measurement testbeds at AVL more flexible. To allow management and reconfiguration within a larger scale the system has to be ARROWHEAD compliant. The system has to be

able to integrate legacy systems, therefore a refit option is necessary.

For distributed measurements to be consistently aligned a global notion of time within the network is necessary. Clock synchronization performance is hampered by delays and jitter accumulated not only in the network, but also in the timestamping procedures of the devices being synchronized. The resources available on such sensors are often constrained, both by cost and power consumption. Software solutions on single processor systems are common. While the synchronization process is critical, it usually has a supporting role and therefore cannot be assumed to have the highest process priority.

To cope with these challenges, the following problems are tackled within this thesis:

- Development of a distributed, wireless measurement system within the ARROWHEAD framework.
- Synchronizing the network, based on IEEE 1588-2008
- Identification, measurement and simulation of clock synchronization error sources.
- Development of strategies to improve clock synchronization.

The proposed implementation assumes a working communication between the network devices. Wireless communications are a lot more sensitive to external interference than wired systems, which may disrupt communication and cause the synchronization to fail. Dealing with external interference is not part of this work, however network access conflicts by nodes on the same network are evaluated. The Arrowhead-Framework was still in development during this work, regular updates (so-called deliverables) were published by working groups around the world. To provide a consistent environment the 1.9.2016 was chosen as a cutoff date after which no deliverables would be incorporated into the test system.

1.2 ARROWHEAD Framework

A proposed solution to the growing complexity problem is the use of a service oriented architecture [Erl08]. This design pattern uses distributed services with loosely coupled interfaces to provide the business logic. The service is the basic building block of the system, and can usually be abstracted by a black box. Services are optimized for reusability instead of being specialized solutions for specific applications. By managing the connections between the services, the system can be reconfigured easily, and new services can be added without interfering with existing applications.

The Arrowhead Framework represents an EU project focused on automation, generating a service oriented reference architecture for interoperability. The goal is to have this reference architecture provide a way for interoperability for common IoT target architectures. Therefore it focuses on local clouds governed through common core services, which may be interconnected for global collaboration (fig. 1.1). The project was started in 2013 and was in development during the

creation of this thesis. In its current implementation this reference architecture should be seen as a proof of concept [VBF+17]. It includes principles on how to design, implement and document SOA, and offers a software framework to support implementations.

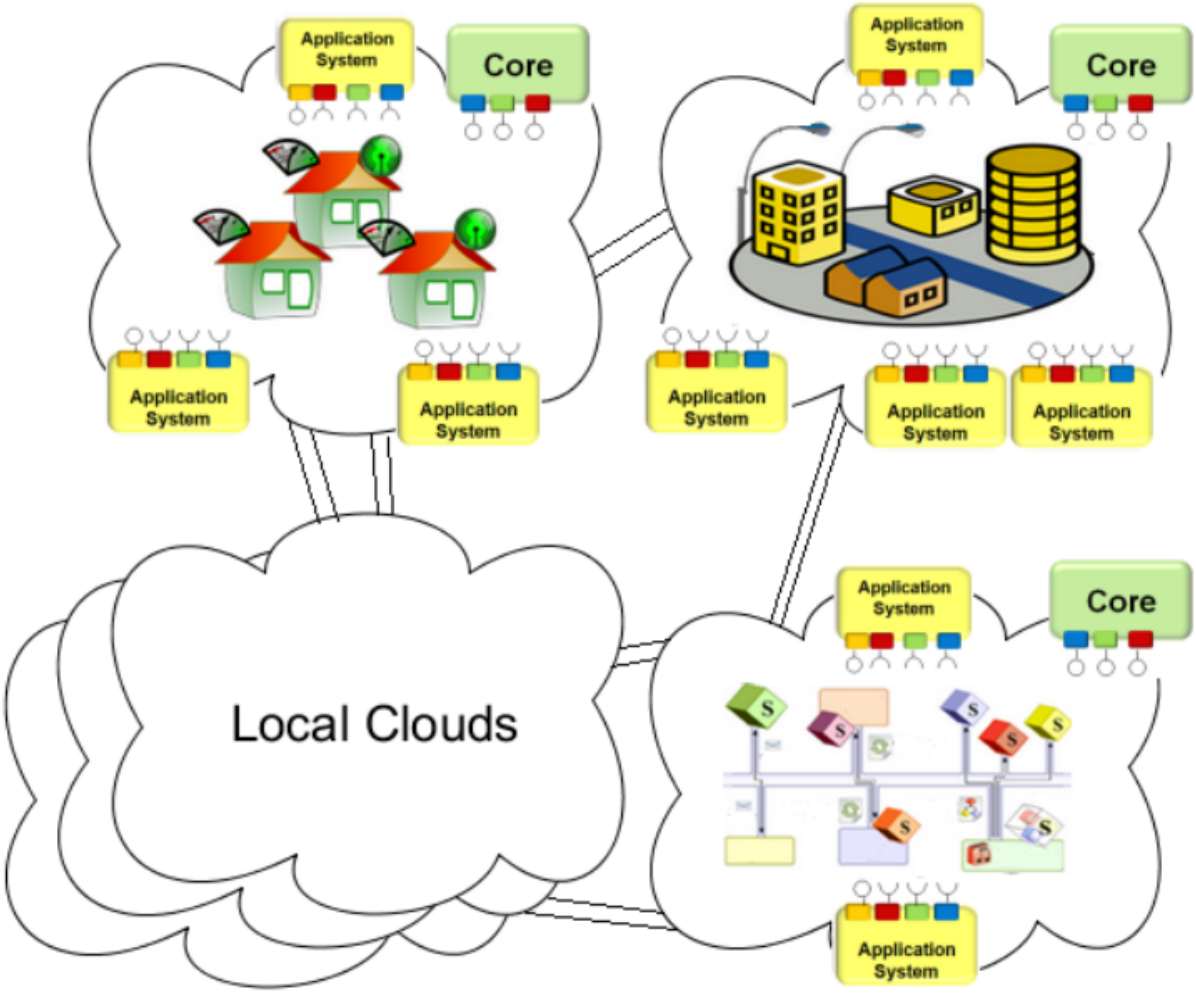


Figure 1.1: Interconnected local collaborative clouds [VBF+17]

A set of core services, which are mandatory for every ARROWHEAD compliant local cloud are defined. These provide any application systems core functionality, providing the basic services necessary for a loosely coupled event based automation cloud.

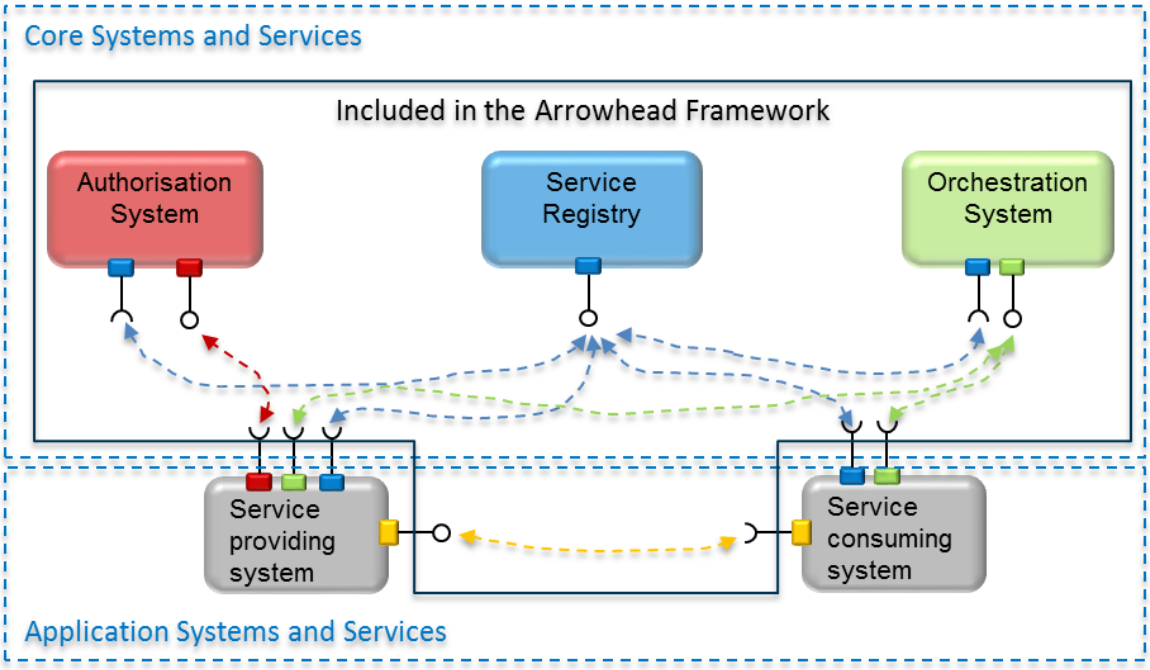


Figure 1.2: ARROWHEAD Core Services [BFK+14]

Service Discovery To be able to dynamically configure a system, it is necessary to be able to find services. This may be either for a management software to tie services together, or for services themselves to search for other services according to their current needs (e.g. a data format conversion service). Therefore the ARROWHEAD framework includes a service registry which allows registration and discovery of services based on various properties. The service registry is based on the DNS-SD system which makes it scalable even on a global level.

Authorization While total flexibility and access are good for a fast dynamic systems, not all actors are beneficial to the system. Especially when systems get connected to the internet security becomes a main concern. Therefore an authorization system is implemented, providing rules and limitations to which services can connect to each other. A service provider can check with the authorization system for any connection requests it receives.

Orchestration After making themselves known to the Service Discovery the application services are initially passive and on stand-by. The orchestration system then connects and configures these services to fulfill a specific functionality. Therefore the Orchestration System establishes all other systems by providing coordination, control and deployment.

To enhance the local cloud, several additional services are considered important optional extensions: Deployment service, User-System Registry service, Configuration service, Event Handler service and Meta-Service Registry service. In addition a set of translation services may be employed, allowing the system to bridge divergent systems (fig. 1.3). Dynamically managing and

cascading these translating services can provide interoperability, with each new translation service added expanding the reach of all others.

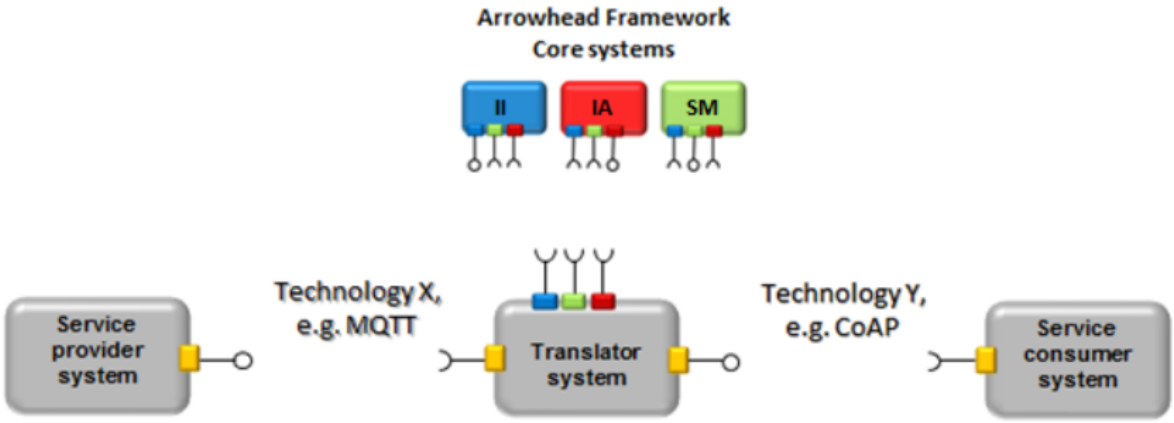


Figure 1.3: Translation as a service [VBF+17]

These core services, coupled with unified documentation standards [BFK+14], ease integration of different systems. Because it would be unrealistic that existing infrastructure would adapt instantly, the ARROWHEAD framework additionally provides design rules for adapters, which can be used to integrate legacy systems into a larger framework.

1.3 Structure of the thesis

Chapter 2 *State of the Art and Related Work* defines terms, provides an introduction into the technologies used and discusses related work of clock synchronization in wireless networks.

In chapter 3 *System Overview* the implementation of the system is presented. The distributed system consists multiple measurement nodes being synchronized by a PTP master node, a local cloud providing core ARROWHEAD services and an orchestration server which uses the ARROWHEAD services to discover measurement nodes and conduct distributed measurements. The measurement application is presented in detail, global design decisions are discussed and all components of the distributed measurement system are described in detail.

This implemented system is the basis for the challenges identified and analyzed in chapter 4 *Challenges of software based clock synchronization*. Here both measured PTP timestamping data and simulations are used to analyze the synchronization offset due to asymmetric packet compression, timestamping Interference from periodic non-interruptable tasks and the influence of network conflicts. An overview how the possible timestamping points influence these kinds of errors is given.

Chapter 5 *Improving software based clock synchronization* uses these analyzes information to deal with these challenges. As these errors cannot be prevented during the timestamping itself they

have to be compensated in postprocessing. This chapter analyzes different filtering strategies to improve clock synchronization, documenting a filter evolution towards the dual drift-compensated uneven median filter. How these filters can be optimized to work efficiently on resource constrained platforms is discussed and the chapter concludes with guidelines how to select appropriate filters for a measurement system.

Implementing these strategies allows for the measurements presented in chapter 6 *Evaluation*. Here clock synchronization accuracy and precision is measured both externally by using an oscilloscope, and internally by performing a synchronized measurement and comparing the measured data. It is shown that synchronization accuracy and precision below $100\mu s$ can be reached in practical applications.

Finally, chapter 7 *Conclusion and Outlook* summarizes this work, provides an overview of the contributions and an outlook on future works.

While this work can be read linearly, sometimes later chapters influence earlier ones. This is especially true of the implemented system (chapter 3), which is influenced by the identified challenges (chapter 4) and uses the filter strategies developed in (chapter 5).

The main contributions of this work can be found in the detailed analysis of the unique challenges of software only timestamping and the development of the dual drift-compensated uneven median filter to deal with these problems. The solutions developed in this work may allow simpler devices to be used in measurement scenarios which previously required dedicated hardware. Because these solutions can be implemented in software this allows portability, reducing both development effort and cost of future devices. It is shown that even with software only timestamping, no tunable clock and interferences from both network and other processes high clock synchronization accuracies are possible. Parts of this work concerning the interference of periodic non-interruptable tasks have already been published in [MSPS17].

2 State of the Art and Related Work

This chapter serves as an introduction to the technologies used and gives an overview of related works. The first section starts by a quick introduction of protocol stack models to put the technologies into perspective and then presents technologies used. Some alternatives, which were considered in this project, are also shortly described. This is followed by an overview of related work, presenting both analysis of the currently available solutions and published different approaches to the problem.

2.1 Basic Technologies

Due to the distributed nature of the implemented measurement system several technology groups need to be introduced. These are wireless network technologies, communication protocols and the main focus of this work, clock synchronization. Working together, these technologies form the basis for distributed, synchronized networks.

2.1.1 Protocol Stack Models

This section covers basic models of how communication between machines works and how it might be implemented. While not directly used in this work, these models provide a reference to understand how the technologies presented interact with each other.

The OSI model

The Open Systems Interconnection model [Sta96], short OSI, is an abstraction model to describe communication between computing systems. Developed by the International Organization of Standardization (ISO), its goal is to provide communication between different systems by using standardized protocols. The model splits the communication process on each host into seven abstraction layers, each one serving the layer above and using the layer below. Outgoing data is encapsulated by each layer until it gets transmitted on the physical layer, and unpacked by the corresponding layer on the receiving host. As each layer is providing abstracted functions, this encapsulation results in each layer communicating directly with its counterpart on the receiving

host, regardless of the protocols used on layers below. This allows for functionality to be split, allowing specialized protocols on each layer which can work together to provide a wide range of functions. The seven layers and their main functions are:

7. *Application* High level application interfaces
6. *Presentation* How data is presented, e.g. character encoding, compression and encryption
5. *Session* Managing continuous information exchange between nodes
4. *Transport* Reliable transmission of data
3. *Networking* Structuring and addressing on a multi-node network or multiple connected networks
2. *Data Link* Reliable data frame transmission between two nodes directly connected via a shared medium
1. *Physical* Transmission of raw bits over a physical medium (e.g. radio waves)

Ideally this would result in protocols which only have to concern themselves with functions on their own layer, and allow flexibility as protocols could be stacked and replaced interchangeably while maintaining well-defined abstract interfaces. In practice, most communication systems do not strictly adhere to this split. The OSI model expects functionality to be handled at specific layers. However, technologies differ in functionality. If functions cannot be provided at the layer they are expected, higher layers either have to provide this functionality themselves or accept the limitation. This results in a technology dependence of the communication protocol stack, preventing a unified model where protocols could be interchanged at will. Nevertheless, the OSI model is widely used as a conceptual blueprint for network architectures and is very useful to classify communication systems and protocols.

Internet protocol stack

The internet protocol stack, defined in RFC 1122 [Bra89] by the Internet Engineering Task Force (IETF), predates the OSI model and forms the basis for the protocols used in internet communication today.

Application layer The top layer of the internet protocol suite without subdivisions, combining layers 6 and 7 of the OSI model. Two categories of application layer protocols exist: user protocols that provide services directly to users and support protocols that provide common system functions.

Transport layer This layer provides end-to-end communication services. The main protocols used are TCP and UDP.

Internet layer The internet layer provides connectionless datagram delivery services by using the Internet Protocol (IP). While other supporting protocols exist for specific functions (e.g. the Internet Control Message Protocol ICMP), all transport layer protocols have to use the IP protocol for transmissions.

Link layer The interface to its directly connected network, wide varieties exist depending on the type of network used.

Even though a physical network is assumed, the IP protocol stack does not concern itself with the technologies used.

The IETF relies more on architectural principles than on strict layering of their protocols. The update RFC 3439 [BM02] includes a section "Layering Considered Harmful", where the limitations of layering are discussed. Strictly adhering to layers results in each layer having to complete its functions before passing data on. Therefore each layer can only be optimized separately. In addition separation may hide information from lower layers which may be needed by those, in turn leading to information or functionality being duplicated. Therefore it may be concluded that horizontal separation (e.g. different protocols for different use cases) may be more cost effective and reliable.

However, in the same document a corollary called "Optimizations Considered Harmful" is introduced, which warns against optimizing systems too much as past a certain point further optimization increases complexity while only providing diminishing returns. Therefore a balance has to be reached.

2.1.2 Wireless network technologies

In this section current wireless networking technologies are discussed. These technologies form the basis for communication between the devices. Technologies discussed are the widely used IEEE 802.11, also known as WiFi, IEEE 802.15.4 which is used in low power sensor networks and bluetooth with a focus on personal area networks.

IEEE 802.11

IEEE 802.11 is a set of standards for wireless local area networks, commonly known as WiFi. The standard is being developed constantly, mainly to provide increased bandwidth. New versions receive a letter combination to distinguish them from previous versions. The current standard rollup, IEEE 802.11-2016 [77816], includes the amendments 802.11 a, b, g, n, ac and ad which define the physical layer and media access control specification for the networks (OSI layers 1 & 2). Frequencies used are $2.4GHz$ and $5GHz$ with data rates up to $866Mbit/s$ and ranges up to $70m$ in an indoor environment, excepting the ad standard which is used for very high data rate ($6757Mbit/s$) low range ($3.3m$) communication at $60GHz$. In recent years, due to developments for smartphones, significant reductions in power consumption were achieved but long term operation of battery powered devices is still not possible.

Usually the network is set up as a star topology with an access point in the center, through which all traffic is routed. Alternatively, peer-to-peer ad-hoc networks are possible which allow mesh routing over multiple hops.

The newest version, IEEE 802.11ah [79217], released in 2017, uses $868MHz$ and $915MHz$ license free bands and is geared toward low-rate, long range communication. Each of its regular channels supports data rates of $100kbit/s$, though data rates up to $347MBit/s$ are possible if a full $16MHz$ channel can be used. With its reduced power consumption and range up to $1km$ this technology may impact the IoT market over the next few years [ABB+14].

IEEE 802.15.4

Low-Rate wireless personal area networks, defined in IEEE 802.15.4 [74616], are used in many low power sensor networks. It defines the encoding and media access control (OSI layers 1 & 2) for radio transmissions at $868MHz$, $915MHz$ or $2450MHz$ at data rates of $20, 40, 100$ or $250kbit/s$. It is the basis for protocols like ZigBee, ISA100.11a, WirelessHART, MiWi, SNAP, Thread and can be used with 6LoWPAN to interconnect with IPv6 networks. Data is encoded either by binary or quadrature phase shift keying. It's design goal is to provide low-cost and low-power communication to a range of $10m$. Multihop peer-to-peer networks are possible, negating the need for additional infrastructure. It does include optional support for real-time communication by reservation of guaranteed time slots.

Bluetooth

Bluetooth is a wireless networking standard introduced by the Bluetooth Special Interest Group in 1994. Its main focus is to be a wireless personal area network, replacing cabling between several devices, rather than a networking standard. Rather than just standardizing the media access control and the physical radio connection, bluetooth standardizes a full protocol stack and several application profiles, therefore covering OSI layers 1-7. Bluetooth uses a packet based protocol in a master slave architecture, with a master controlling a synchronized network of up to 7 slaves. To avoid interference it uses frequency hopping, using 80 $1MHz$ channels in the $2.45GHz$ band at 800 hops per second, reaching an application data rate of up to $2.1Mbit/s$. Range is defined by the transmission class of the host, with class I reaching up to $100m$.

IEEE has standardized Bluetooth 1.2 as IEEE 802.15.1 [14905], newer versions however are only standardized by the bluetooth special interest group. Due to different applications several variants are available. Introduced in Bluetooth 3.0 a high-speed link allows for fast data transfer. The system uses bluetooth for negotiation and establishment of a 802.11 link for the actual data transfer of up to $24Mbit/s$. In contrast, Bluetooth Low Energy, introduced in version 4, uses a reduced data rate of $270kbit/s$, combined with faster wakeup times and an optimized protocol stack to drastically reduce power consumption [GOP12].

2.1.3 Communication Protocols

Wireless technologies allow devices to communicate with each other, protocols give this communication of devices structure. The protocols used for machine to machine communication in this work are briefly explained.

IPv4 & IPv6

The Internet Protocol is a communication protocol for relaying packets across network boundaries. It encapsulates the data with a header with containing source and destination addresses, which allow for delivery in a worldwide network. Its first mayor release in 1981, the Internet Protocol version 4 [Pos81a], defined a connectionless protocol for best-effort delivery on packet-switched network. It does not provide reliability, but it provides very robust and flexible routing within its 32 bit address space. Its most recent version, the Internet Protocol version 6, uses 128 bit addresses, and was developed by the IETF when it became apparent that the addresses available in IPv4 were not sufficient for the rapid growth in devices connected to the internet.

Besides the larger address space IPv6 supports hierarchical address allocation which reduces routing complexity, improves multicasting, allows for stateless address autoconfiguration and allows for simplified extensions using extension headers. Packet fragmentation was removed, as fragmentation and reassembly are a big source of overhead in IPv4. Instead it requires the maximum transmission unit to be at least $1280 Bytes$, and has mechanisms to detect the MTU of a connection before data transmission begins.

6LoWPAN

6LoWPAN is the abbreviation of IPv6 over Low-Power Wireless Personal Area Networks, and is also designed by the IETF. It was developed to allow IPv6 use in low power devices, which usually have less processing power and are using low bandwidth connections. The standard assumes usage on IEEE 802.15.4 wireless networks. Due to the limitations of IEEE 802.15.4, most notably its frame size limit, several modifications have to be made:

Packet fragmentation — Due to the $127 Byte$ frame size limitation of IEEE 802.15.4, standard IPv6 packets with a minimum size of $1280 Bytes$ cannot be transmitted in single frames. Therefore the minimum packet size is dropped and 6LoWPAN implements an adaption layer for packet fragmentation and reconstruction to handle bigger packets.

Header compression — To improve efficiency, 6LoWPAN offers optional header compression. This is necessary as a full $40 Byte$ IPv6 header can use over 30% of a frame, a number that even increases when higher-layer headers are factored in. The compression methods used can reduce the IPv6 header down to $2 Bytes$ in the best case, and can additionally compress the predominant OSI-layer 4 protocols UDP, TCP and ICMP.

Address autoconfiguration 6LoWPAN also defines how the stateless address autoconfiguration IPv6 addresses are generated from the IEEE 802.15.4 interface addresses. Defining this algorithm aids in address compression, as redundant information from frame addresses can be avoided.

In addition, the 6LoWPAN protocol stack is also optimized for code size, energy efficiency and routing in mesh topologies.

RIME

The RIME communication stack [DÖH07] consists of a set of communication primitives. It's goal is to completely abstract the network technologies used from the application to allow for ease of portability. Its primitives consist of several methods for packet transmission including unicast, multicast and broadcast, each consisting of several variants like best-effort, reliable, identified and polite. Depending on the underlying architecture the RIME protocol stack it either uses the underlying protocols to provide these functions, or it emulates them. Therefore the RIME stack cannot be easily placed on the OSI-model as it can provide layers 2-5, but may also reduce itself to upper layers if the methods necessary are provided by underlying protocols. It does not define headers like most communication protocols, instead packets receive a set of attributes in addition to their data.

Underlying the RIME primitives are the chameleon header transformation modules. These depend on the network implementation and are tasked with generating appropriate headers from the packet attributes provided. With these abstraction layers communication can be abstracted from the network itself, allowing applications on heterogeneous wireless sensor networks to communicate with each other.

TCP

The Transmission Control Protocol [Pos81b] is used to provide reliable data transfer on connectionless networks. Before data communication can occur, connections have to be established using a three-way handshake. After this the connection is considered open, local buffers are allocated and data transfer can commence. All TCP messages carry a sequence number, and uses acknowledgments to signal received packets. Should an acknowledgment fail to arrive after a timeout, data packets automatically get retransmitted. TCP additionally provides flow and congestion control, to prevent faster connections to overwhelm slower ones. This is done by two mechanisms: to provide flow control the receiver transmits its window size, the number of bytes it can currently accept within its buffer. Once this window is full the sender stops transmitting until it receives an acknowledgment with a new window size from the receiver. To avoid congestion on the network between the sender and the receiver the data rate may also be throttled if acknowledgments fail to arrive. This is to avoid congested networks, where no control messages could be transmitted between sender and receiver.

To identify sending and receiving applications TCP uses port numbers. Each communicating application associates with (at least) one local 16-bit port. Connections are identified by their sockets, which is the combination of the source host address, source port, destination host address and destination port. This allows a host to provide several services simultaneously, and distinguish between different connections to the same service.

A typical TCP header uses $20 Bytes$ of data. This can be extended by the variable length option header up to a total length of $60 Bytes$, however these optional extensions are rarely used.

UDP

The User Datagram Protocol [Pos80] is a simpler protocol for transmissions. Similar to TCP it provides port numbers (and therefore sockets) and checksums, but no handshake or congestion control. It is designed to limit overhead for applications where fast delivery and efficiency are more important than reliable data transfer. Therefore it only needs $8 Bytes$ for its header.

COAP - Constrained Application Protocol

The Constrained Application Protocol is an application layer protocol for resource constrained devices, standardized in RFC 7252 [SHB14]. It is similar to HTTP which is used for the world wide web, however it has been optimized for a very small footprint and less resource usage, both of processing power and bandwidth. To save on both it is based on the connectionless UDP transport layer protocol, to balance resource use with reliability it contains both confirmable (which elicit acknowledgment messages) and non-confirmable messages.

COAP allows access to resources which are identified by an URI (uniform resource identifier). If only the host is known available resources can be discovered by accessing the defined resource /.well-known/core in the CoRE link format [She12].

It's main methods for interacting with resources are

GET The GET method retrieves a representation of the resource specified in the URI.

POST The data attached in a POST message gets handled by the specified resource. How that data gets handled depends on the resource itself.

PUT This method creates a new resource at the URI, or modifies an existing one.

DELETE DELETE requests for the specified resource to be deleted.

OBSERVE This optional extension added in RFC 7641 [Har15] greatly reduces overhead for polling changing resource. If a resource gets observed data gets sent automatically to the observing host. These can happen either on an event-basis (e.g. triggered by changing resources) or time-based (e.g. regular intervals).

These methods elicit response codes consisting of 3 digits (e.g. 200 OK), which can be piggybacked onto responding data messages. The functions GET, POST, PUT and DELETE can be mapped easily to HTTP which allows for interoperability with HTTP resources. By using proxies it is possible for COAP hosts to interact with HTTP hosts while locally retaining the benefits of the reduced resource usage (fig. 2.1).

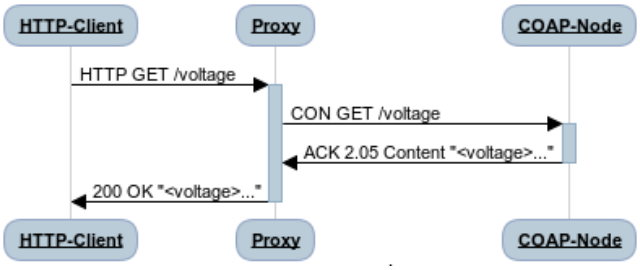


Figure 2.1: COAP to HTTP communication with proxy

OMA-LwM2M

The Lightweight Machine-To-Machine protocol of the Open Mobile Alliance (OMA-LwM2M, [Tia12]) is a device management protocol. It defines a communication protocol between a LwM2M server and LwM2M clients, usually resource constrained IoT devices. Clients register themselves at a LwM2M server, and can be monitored and managed from this central location. The LwM2M protocol is built upon COAP and uses a standardized, but extensible, resource model for device monitoring and configuration. It is optimized for cellular or sensor networks. Several open source implementations, so called enablers, exist and are free to use.

2.1.4 Clock Synchronization

The goal of clock synchronization is to have a single global time-base for all involved nodes. This is especially important for distributed measurement systems, as data from multiple devices needs to be aligned properly to be able to analyze the data. Without proper clock synchronization it is impossible to tell if an event on another device happened before, after or at the same time as an event measured locally. Clock synchronization protocols specify how devices exchange messages to synchronize their independent clocks.

A clock typically uses an oscillator as a source of a constant frequency which drives a counter. Ideally the frequency of this oscillator would be constant over an infinite time. However the frequency real oscillators like quartz crystals may change due to influences like temperature or aging. If multiple clocks are considered it can also not be assumed that they started counting at the exact point in time. Therefore a clock t_j in a network, in regards to a reference clock t, can be described by

$$t_j(t) = a_j * t + b_j \tag{2.1}$$

As the goal of clock synchronization is to match clocks as closely as possible to a reference clock, an error term which should be minimized can be defined:

$$\epsilon_j(t) = t_j(t) - t \tag{2.2}$$

Important terms and properties include, but are not limited to,

Accuracy The mean difference between the local, synchronized clock and the reference clock ϵ_j. The desired accuracy is dependent on the application. Measurement of slowly changing properties (e.g. weather temperature) might only need a synchronization accuracy of seconds, while measurements of fast changing processes (e.g. optical effects) might need accuracies in the nanosecond range or even better.

Precision A measure of the variability of the clock synchronization, defined as the standard deviation of the error ϵ.

Jitter One of the main challenges of clock synchronization protocols. Due to any number of influences (message delay, different routing paths, limited timestamping accuracy, etc.) not all synchronization messages take the same amount of time from one host to another.

Clock drift All clocks have a limited accuracy. Even if at one point two independent clocks are perfectly synchronized, if one clock ticks slower than another, the two clocks slowly drift apart. Therefore continuous synchronization is necessary.

Precision Time Protocol

The precision time protocol, also called IEEE 1588, is used to improve clock synchronization accuracy in local networks. Two standards exist: PTP version 1 (IEEE 1588-2002, [IS02]) and the improved PTP version 2 (IEEE 1588-2008, [IS08]). Nodes can be connected in an arbitrary topology. Neighboring nodes form a master-slave relationship with a master distributing its clock to any number of slaves. A node can fulfill both roles at the same time, synchronizing itself to its master while distributing its clock to its slave nodes. This leads to the clock being distributed in a tree topology, with a so-called grandmaster clock at the top.

The protocol consists of two parts:

- The Best Master Algorithm is used on every sub-network to select the most accurate clock as the master clock.
- The Clock Synchronization, where messages are exchanged to estimate the delay on the transmission and to accurately synchronize the slave clock to the master clock.

The highly improved accuracy comes from the fact that transmission delays are measured both ways, which allows the protocol to compensate all symmetric delays. It is also often combined with hardware timestamping for even higher accuracies. PTP version 2 comes with several optional annexes, for example to compensate for clock speed differences or handle security issues. Accuracy

has additionally been improved by the introduction of transparent clocks, network devices which measure packet reception and transmission timestamps and include a correction term into the transmitted packets. This allows to compensate for the delays incurred by network devices like routers or bridges. However, the security implementation (IEEE 1588-2008 Annex K, [IS08]) has several compromising weaknesses [TH09][MLJ+15], which are one of the main reasons for the currently developing PTP version 3 standard.

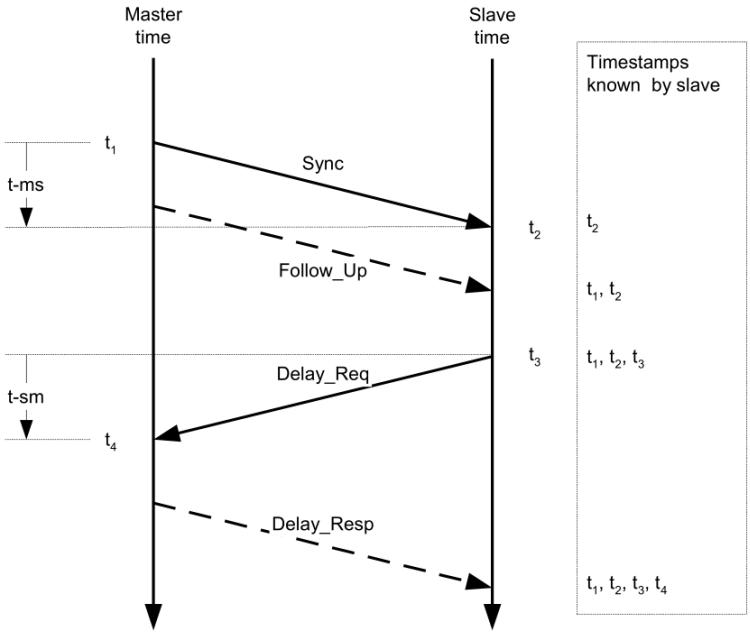


Figure 2.2: PTP synchronization messages, [IS08]

The synchronization process can be seen in figure 2.2:

1. The PTP-Master sends a SYNC message, which gets timestamped on reception by the PTP-Slave. The message includes a sending timestamp. However, unless special hardware is used that can change messages while transmitting, this included timestamp is set by software in higher layers and therefore may be inaccurate. A more accurate timestamp may be taken on transmission and sent with the FOLLOWUP message.

2. The PTP-Master sends a FOLLOWUP message (optional). The message includes the accurate timestamp of the previous SYNC message. This message type is necessary as an accurate sending timestamp can only be taken while sending the SYNC message, and to change the data in the timestamp field while during transmission is only possible with special hardware.

3. The PTP-Slave sends a DELAY-REQUEST message to the PTP-Master. This message gets timestamped both by the PTP-Slave on transmission and the PTP-Master on reception. No FOLLOWUP is needed for DELAY-REQUEST messages as it is not necessary for the PTP-Master to know the precise origin timestamp of the message, however the accurate sending timestamp has to be stored locally.

4. The PTP-Master sends a DELAY-RESPONSE message to the PTP-Slave. This message includes the reception timestamp of the DELAY-REQUEST message.

After these four messages, assuming symmetric delays, the slave has all the information to calculate the offset of his clock to the PTP-Master clock. The local clock of the slave can be expressed by a yet unknown offset to the clock of the master. All timestamps on the slave side have to consider this offset.

$$t_{Slave} = t_{Master} + offset \tag{2.3}$$

The reception timestamp of the SYNC message includes both the offset and the transmission delay.

$$t_2 = t_1 + offset + delay \tag{2.4}$$

With the origin timestamp included the SYNC message or the FOLLOWUP message the sum of the offset and the delay can be calculated. As the transmission delay cannot be negative this provides boundaries for the offset, however no exact offset estimation can be done yet.

$$\Delta t_{SYNC} = t_2 - t_1 = offset + delay \tag{2.5}$$

The DELAY-REQUEST packet gets timestamped both by the slave and the master. As the transmission is from the slave to the master, the offset has to be substracted from t_3 to be in the frame of reference of the master (2.3).

$$t_4 = t_3 - offset + delay \tag{2.6}$$

After the DELAY-RESPONSE message the slave knows all the timestamps needed to calculate its offset to the master. Because the offset effects the transmission delays Δt_{SYNC} and $\Delta t_{DELAY-REQ}$ with different polarities but the delay stays positive the exact offset and the transmission delay can be calculated.

$$\Delta t_{DELAY-REQ} = t_4 - t_3 = -offset + delay \tag{2.7}$$

$$offset = \Delta t_{MS} = \frac{1}{2} * (\Delta t_{SYNC} - \Delta t_{DELAY-REQ}) \tag{2.8}$$

$$delay = \frac{1}{2} * (\Delta t_{SYNC} + \Delta t_{DELAY-REQ}) \tag{2.9}$$

with

t_{Slave} The local clock time of the PTP-Slave

t_{Master} The local clock time of the PTP-Master

$offset$ The clock offset between PTP-Master and PTP-Slave

$delay$ The transmission delay between PTP-Master and PTP-Slave

t_1 The transmission timestamp of the SYNC message, by the PTP-Master clock

t_2 The receiving timestamp of the SYNC message, by the PTP-Slave clock

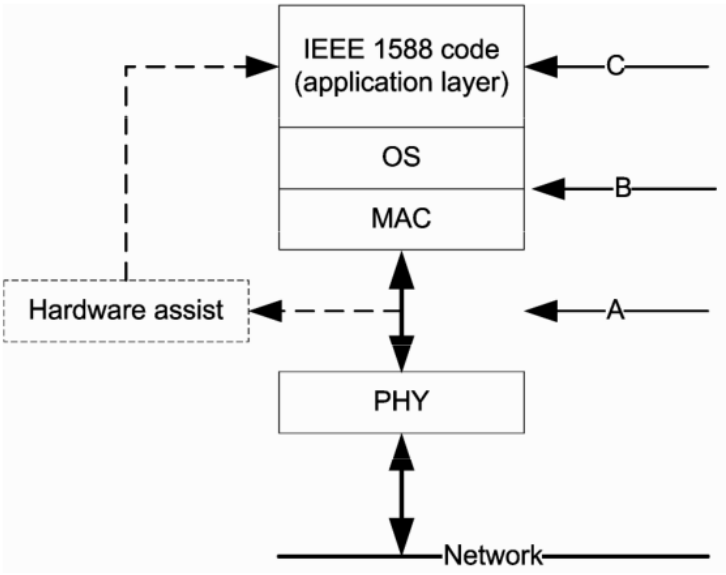


Figure 2.3: PTP message timestamp points [45708]

t_3 The transmission timestamp of the DELAY-REQUEST message, by the PTP-Slave clock

t_4 The receiving timestamp of the DELAY-REQUEST message, by the PTP-Master clock

However, this assumes that the delay is both symmetric and constant, and that both clocks run at exactly the same frequency. Due to limited accuracy and changing conditions like transmission interference this assumption does not hold true. This is especially problematic if the timestamping cannot be done by specialized hardware on the physical layer (point A in fig. 2.3), but has to be done at higher layers. Chapter 4 analyzes the challenges that arise from timestamping by software at point B, while chapter 5 provides filter strategies to keep the synchronization accuracy high.

Network Time Protocol

The Network Time Protocol (NTP) is designed to synchronize computer systems over variable latency packet switched networks. Its goal is to synchronize all hosts to the Coordinated Universal Time. Its current version, NTP v4, is standardized in RFC 5905 [MMBK10]. However it is designed to be backwards compatible to NTP v3.

Time servers across the world are available to accomplish this task, organized into a hierarchical layout segmented into strata. At the top strata 0 high precision devices like atomic clocks or GPS synchronized devices. Directly attached to them are primary time servers at stratum 1. It would be impossible to let the whole world synchronize directly from these devices, therefore each following strata synchronizes with the clocks above it, distributing the load and providing globally distributed NTP servers.

The protocol uses a client-server model. Round trip delay measurements, timestamped via software, to multiple clock servers are measured. Outliers are discarded and the time offset to the server estimated from the three best sources. A modified version of Marzullos algorithm [MO83] is used to calculate the most likely current time and estimates the confidence interval. It does this by analyzing the overlap of the clock results and their respective confidence intervals from each NTP server, looking for intervals which are consistent with the largest number of sources.

NTP provides clock synchronization accuracies around $10ms$ over the internet, and can achieve sub-ms accuracies on local networks.

2.2 Related Work

Due to limited availability of solutions for the specific scenario, related work is presented in two parts:

IEEE 802.11 Clock Synchronization	IEEE 802.11 is the most prevalent wireless network technology used, therefore a lot of work has been done on this technology. These solutions often offer clock synchronization with software only timestamping, however due to the network technology the focus is on performance instead of power efficiency.
Wireless Mesh-Network Clock Synchronization	These are optimized solutions for wireless mesh networks, developed either for the IEEE 802.15.4 network used in this work or based on similar technologies.

2.2.1 IEEE 802.11 Clock Synchronization

While using a different base technology than this work, IEEE 802.11 wireless networks includes most of the same basic challenges. As 802.11 is widely used both in industry and consumer areas, more work exists on clock synchronization in these networks. [METS17] provides a recent survey over clock synchronization protocols and methodologies.

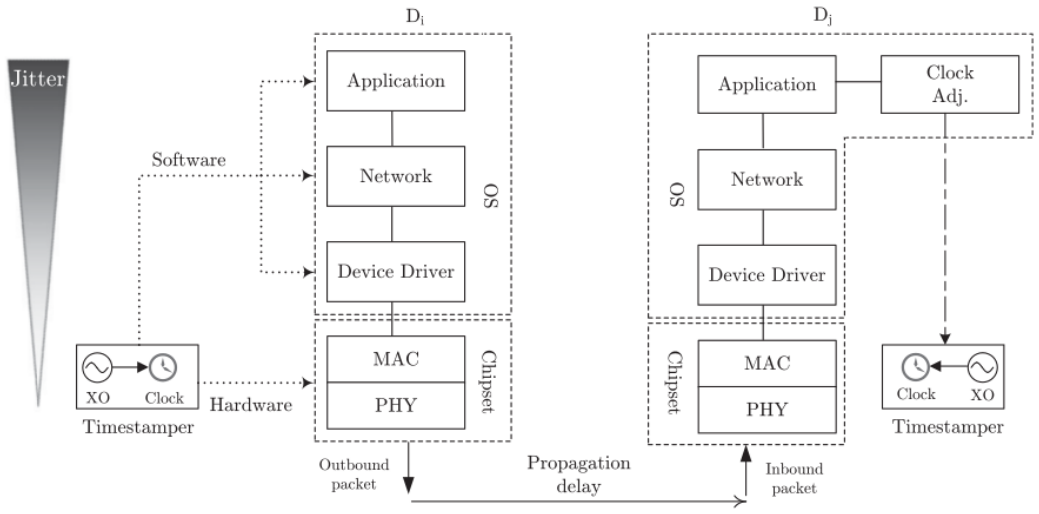


Figure 2.4: Various factors affecting packet-based synchronization [METS17]

Figure 2.4 shows the path a synchronization packet has to take in a sender-receiver clock synchronization protocol. The possible timestamping points for outbound packets are shown. As the sources of jitter are cumulative timestamping farther from the physical transmission greatly increases jitter. However it can also be seen that one source of jitter, the oscillator itself, influences all timestamping points.

The evaluated protocols are classified by several characteristics:

Architecture Whether the system uses a centralized clock source or all clients synchronize with each other using a peer-to-peer synchronization protocol.

Operation mode Depending on which node initiates the clock synchronization protocols can be classified as Client-Server or Master-Slave protocols.

Origin Protocols which were developed for wired networks and ported to IEEE 802.11 or protocols which were developed for the wireless technology.

Clock source Clocks can either be synchronized either to an absolute clock source such as the Universal Coordinated Time (UTC) or just use relative synchronization to provide a synchronous network time.

Operation principle Sender-Receiver protocols synchronize their clocks to a reference node which sends synchronization messages, while Receiver-Receiver protocols timestamp received broadcast messages and then exchange their information to synchronize with each other.

Table 2.1 shows the result of the survey of current protocols. These protocols are grouped as either being native to/part of IEEE 802.11, or into generic protocols which have been implemented on this technology. The solutions included in the IEEE 802.11 standard used to be solely for physical and MAC layer operations, however they can be accessed by higher layers since IEEE 802.11-2012. They offer high accuracy due to hardware timestamping, however they do not compensate

Table 2.1: Summary of Clock Synchronization Solutions for IEEE 802.11 [METS17]

Description	HW/SW	Propagation Delay Calc.	Over-head	Accuracy	Precision
IEEE 802.11-based solutions					
TSF Scheme	HW	None	None	$4.0\mu s$	$< 1.0\mu s$
Timing Measurement (TM) method	HW	Two-way	None	NA	NA
Timing Advertisement (TA) method	SW	None	None	$0.50\mu s$	$2.50\mu s$
non-IEEE 802.11-based solutions					
TSF with virtual clocks	SW	None	None	$10.0\mu s$	NA
SyncTSF	SW	None	None	$1.8\mu s$	$0.5\mu s$
PTP using Windows driver	SW	Two-way	Yes	$5.5\mu s$	$6.3\mu s$
PTP using Linux driver (1)	SW	Two-way	Yes	$2.3\mu s$	$4.4\mu s$
PTP using Linux driver (2)	SW	Two-way	Yes	$6.6\mu s$	$0.59\mu s$
PTP with optimized parameters in Linux	SW	Two-way with in-device delay compensation	Yes	$59ns$	$0.46\mu s$
PTP with HW timestamping (1)	HW	Two-way	Yes	$1.1ns$	$3.1ns$
PTP with HW timestamping (2)	HW	Two-way with PHY delay compensation	Yes	$240ps$	$531ps$
NTP with application timestamping	SW	Two-way	Yes	$0.5ms$	$2.39ms$
NTP with driver timestamping	SW	Two-way	Yes	$2.72\mu s$	$5.27\mu s$
Application CS with Beacons	SW	None	None	NA	$150\mu s$
LS Algorithm for CS	SW	Two-way	Yes	$35\mu s$	$15\mu s$
Infrastructure mode-based RBS	SW	None	Yes	$0.2\mu s$	$0.18\mu s$

for any delays incurred on the transmission path. Hardware PTP timestamping offers the highest accuracy by far, offering synchronization accuracies and precision at least one order of magnitude better than all other solutions. Optimizing solutions so in-device errors can be compensated also improves performance greatly, as the Linux implementation shows. However this requires exact measurements of these errors, which make these solutions very hardware dependent.

2.2.2 Wireless Mesh-Network Clock Synchronization

Several specialized clock synchronization protocols exist for wireless mesh networks. In addition to the synchronization they manage the distribution of the clock in the network, as it is assumed that not all nodes are within communication range of each other. This is especially necessary when dealing with geographically spread out, or extremely low power sensor networks. To save power, some protocols do not synchronize nodes until it is needed, sometimes using post-facto synchronization after an event occurs to determine their relative offsets during measurements. Table 2.2 gives a short overview over different synchronization protocols.

Table 2.2: Overview over wireless synchronization protocols (Expanded from [RAK10])

Name	Accuracy	Basic Method	Compensates	HW/SW
TPSN [GKS03]	$16.9\mu s$	Round Trip Delay in Sync Tree	Offset only	SW
TS/MS [SV03]	$945\mu s$	Round trip delay and line fitting	Offset & Rate	SW
RBS [EGE02]	$1.85\mu s$	Reference broadcast detection	Offset & Rate	SW
TDP [AS05]	$100\mu s$	Round trip delay measurement	Offset only	SW
Mod. PTP [WM10]	$10\mu s$	Round trip delay measurement	Offset only	HW

Time Synchronization Protocol for Sensor Networks

The Time Synchronization Protocol for Sensor Networks (TSPN), introduced in [GKS03] assumes that resource constricted nodes are arranged in an extended wireless network. A symmetric link is assumed. Direct link quality between nodes may vary and therefore multihop communication may be necessary. Therefore it builds a synchronization tree starting at a predetermined root node before using pairwise synchronization along that tree to synchronize nodes. Only offset correction is performed, clock rate errors which are low enough to prevent serious drifting of nodes within the synchronization interval are assumed.

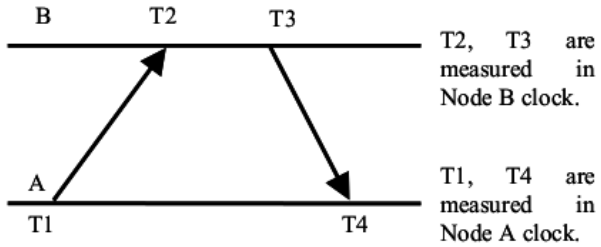


Figure 2.5: Two-way message exchange between pair of nodes [GKS03]

The protocol works in two steps:

Level Discovery Phase The root node broadcasts a level discovery packet, at his own level 0. Neighbors which receive a level discovery packet note the functioning connection, assign themselves to the received level plus 1, and broadcast level discovery packets themselves. Further received level discovery packets are ignored. This leads to a flooding of the network until every node has a level and knows its nearest parent node.

Synchronization Phase In the synchronization phase each node regularly synchronizes itself with its parent node. This is triggered by the parent node sending a TIME SYNC broadcast. After a random delay to avoid congestion every child node initiates a two way message exchange of synchronization messages. This results in a round trip delay measurement to the parent nodes 2.5. The offset to the parent node can be calculated according to 2.10.

$$offset = \Delta t_{MS} = \frac{1}{2} * ((T2 - T1) - (T4 - T3)) \tag{2.10}$$

Ganeriwal et al implemented this protocol on Berkeley Motes, a wireless node optimized for experiments, using a modified TinyOS to access a $4MHz$ timer and timestamping sent and received packets in software. Multi-Hop networks up to 5 hops were analyzed, leading to the results in table 2.3 over 100 measurements. While one would assume that the clock synchronization accuracy would linearly increase with every hop, that is not the case. Because synchronization errors between pairs are derived from a normal distribution, on average synchronization errors of opposite polarity can compensate each other. This is even more pronounced with errors due to clock rate errors, as all clock quartzes are within a limited range of the base value. Therefore, excepting quantization errors due to clock resolution, all errors due to intermediate clock rates even themselves out. Only the clock rate errors due to the first and the last node remain. Worst case synchronization results however do increase, as errors due to other sources can get compounded.

Table 2.3: Statistics of synchronization error over multihop (only magnitude) [GKS03]

	1 hop	2 hop	3 hop	4 hop	5 hop
Average error	$17.61\mu s$	$20.91\mu s$	$23.23\mu s$	$21.436\mu s$	$22.66\mu s$
Worst case error	$45.2\mu s$	$51.6\mu s$	$66.8\mu s$	$64\mu s$	$73.2\mu s$
Best case error	$0\mu s$	$0\mu s$	$2.8\mu s$	$0\mu s$	$0\mu s$

TinySync / MiniSync

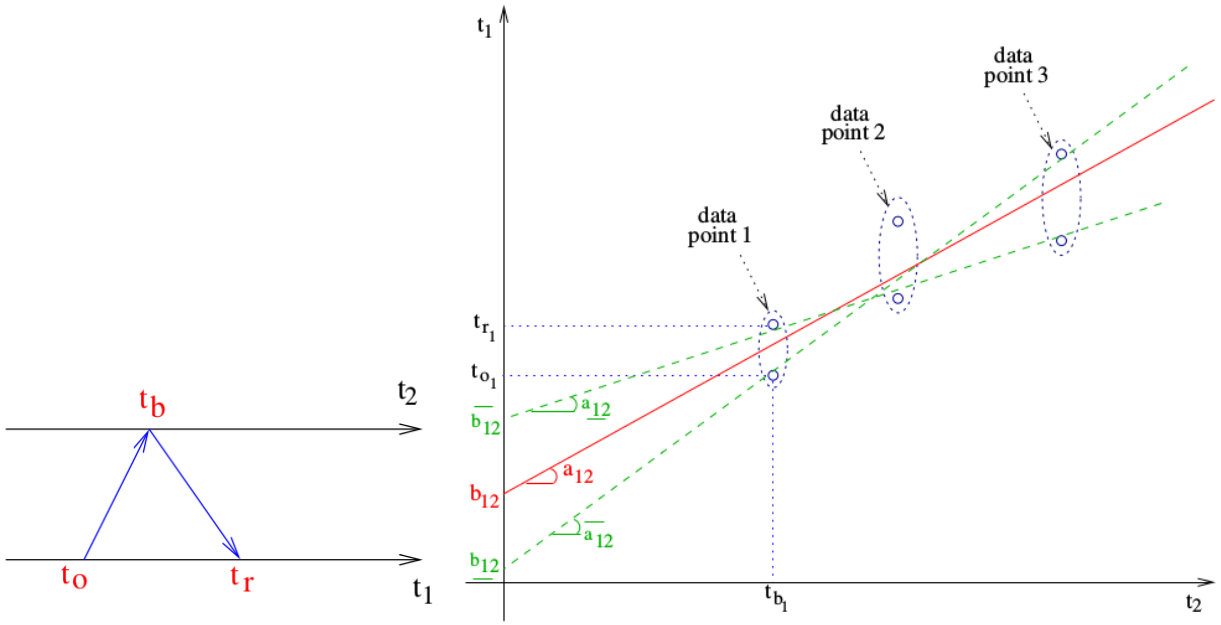
TinySync and MiniSync are two variants of the same clock synchronization algorithm presented in [SV03]. It's goal is to provide deterministic bounds on any clock synchronization, even when used in networks with highly variable packet transmission times and asymmetric transmission paths. The basic method for clock synchronization is offset line fitting over delay measurements, automatically compensating a linear clock rate error in the process. It assumes that a sensor node has limited resources which prevent the calculation using all known measurements. Due to the limited amount of transmitted information and no requirements on periodic synchronization packets, both TinySync and MiniSync may be piggybacked on application traffic. While a hierarchical network for multihop clock synchronization similar to TSPN is used for extended networks, no protocol for establishing this hierarchy is included in the base protocol.

To synchronize its clock a probe message is sent from the child node to its parent, which immediately replies with its current timestamp (fig. 2.6a). The child node timestamps both the sent and the received packet. This creates a set of three timestamps t_o, t_b, t_r for each datapoint. As these timestamps can only be created in order a set of inequalities is created.

$$t_o(t) < a_{12} * t_b(t) + b_{12} \tag{2.11}$$

$$t_r(t) > a_{12} * t_b(t) + b_{12} \tag{2.12}$$

While the exact values of a_{12} and b_{12} cannot be determined, by using line fitting algorithms on multiple measurements (fig. 2.6b) upper and lower bounds for each value can be established.



(a) Clock synchronization **(b)** Linear dependence

Figure 2.6: Probe message from node 1 is immediately returned and timestamped (a), Linear dependence and constraints imposed on a_{12} and b_{12} by three data points (b) [SV03]

As solving these inequations for every datapoint is very computationally intensive TinySync and MiniSync offer different optimizations to this algorithm. TinySync only keeps the two datapoints which offer the most accurate results in memory, all other datapoints are discarded. This dramatically reduces memory usage and computing power. However, just because a datapoint does not improve accuracy with the currently available data does not mean that it may not improve accuracy with future timestamps. Therefore the MiniSync algorithm only discards datapoints which do not contain any additional information and uses line-fitting on the best available set of 2 measurements. Additionally, big improvement to both algorithms is to compensate the minimum delay. If for example the minimum packet transmission time is known periods where the packet could not have arrived at the other node can be defined. By removing minimum delay from the uncertainty window its size is reduced dramatically, improving synchronization accuracy almost by a factor of 5.

Experiments were carried out on a 802.11b multihop ad-hoc network, using 256$Byte$ packets to mimic piggybacking. Over 5000 measurements it has been shown that while suboptimal, after an initial uncertainty TinySync results are within 0.2% of MiniSync for a single-hop environment (1.8% for 5 hops). Table 2.4 shows the final uncertainty for both a_{12} and b_{12}. It has however to be noted that this is only the maximum possible error, unfortunately actual synchronization accuracy was not measured.

Reference Broadcast Synchronization

Jitter of synchronization packets can be generated at three sources: at the sender, on the medium

Table 2.4: Results for TinySync with and without data preprocessing [SV03]

	Raw Data		Minimum delay removed	
	$\frac{\Delta a_{12}}{2}$	$\frac{\Delta b_{12}}{2}$	$\frac{\Delta a_{12}}{2}$	$\frac{\Delta b_{12}}{2}$
one hop	$7.133e-07$	$2.0941ms$	$2.768e-07$	$0.9457ms$
five hops	$5.013e-6$	$17.08ms$	$1.167e-06$	$3.239ms$

and at the receiver. Reference Broadcast Synchronization (RBS, [EGE02]) fundamentally differs from traditional sender-receiver clock synchronization protocols in the fact that it synchronizes a set of receivers with one another. This completely removes the sender as a source of jitter. The RBS concept requires that at least 3 nodes share a broadcast domain, it does not work on point-to-point connections.

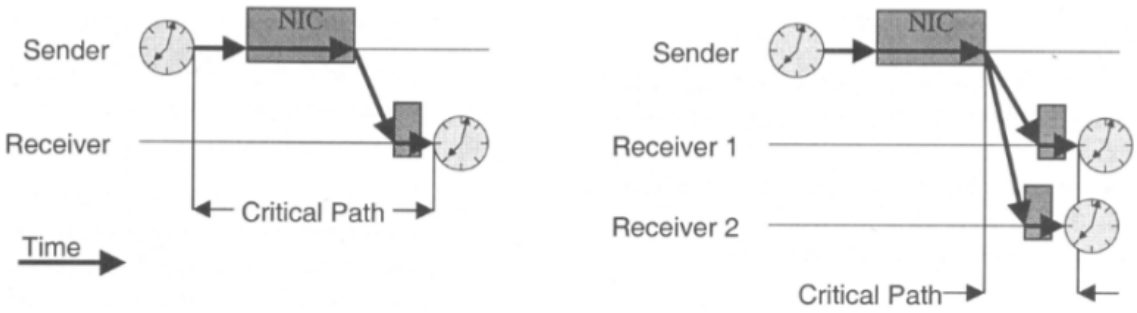
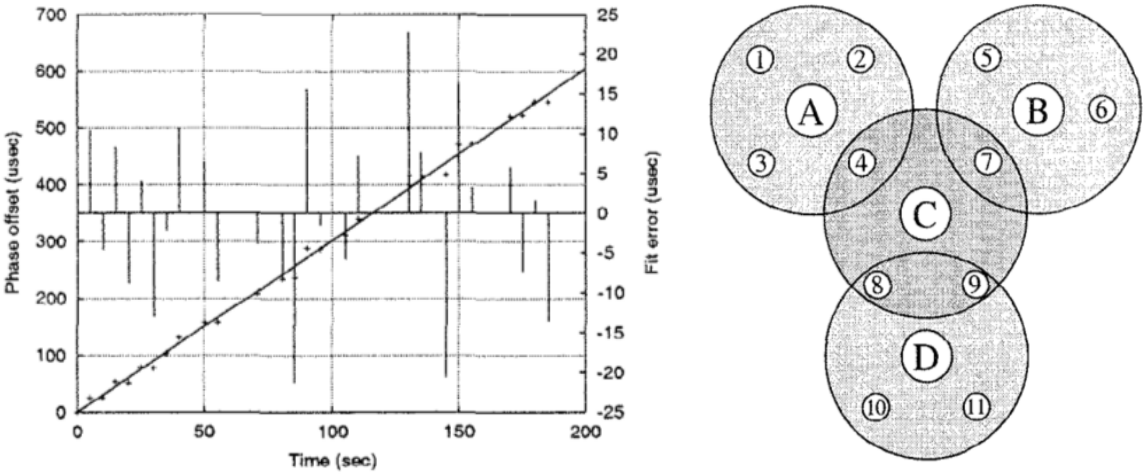


Figure 2.7: A critical path analysis for traditional clock synchronization protocols (left) and RBS (right) [EGE02]

The main challenge for clock synchronization is non-determinism. There are four sources of message latency, which together form the critical path: send time, access time, propagation time and receive time. RBS dramatically shortens the critical path by having nodes periodically send broadcast packets which get timestamped by all receivers on reception 2.7. These packets do not have to contain timestamps. Afterwards the receivers share their local reception timestamps. Limiting the timestamping to reception removes the send time and the access time as sources of non-deterministic errors. On a shared medium propagation speed can be assumed to be very high (c for wireless links, $\frac{2}{3}c$ for wired connections), which means that even multipath propagation will not result in significant jitter. Therefore only the receive time remains as the main source of jitter.

With the information of multiple such broadcasts this allows nodes to calculate their offset and their clock rate difference to all other nodes in the broadcast domain. This was done by calculating the best linear fit line of all the available data after rejection of outliers. This is shown in fig. 2.8a by using $x = T_{r1,k}$ and $y = T_{r2,k} - T_{r1,k}$ for each broadcast. By timestamping the packets on Berkeley Motes in the reception interrupt RBS reached accuracies of $11\mu s$. However only the timestamp collection was done on the nodes itself, the computation for the clock synchronization was done in simulations afterwards. By using multiple broadcasting nodes this method can be extended to multihop networks (fig. 2.8b).

To prove the advantages of RBS are not dependent on the platform used Elson et al implemented RBS in Linux, allowing for comparison with NTP. The timestamping itself was done at the



(a) Clock synchronization **(b)** Multihop topology

Figure 2.8: An analysis of clock rate effect on RBS. Each point represents the phase offset between two nodes as implied by the value of their clocks after receiving a reference broadcast. (a), a more complex 3-hop multihop network topology (b) [EGE02]

Table 2.5: Synchronization error for RBS and NTP between two Compaq IPAQ using 802.11 [EGE02]

Load	Protocol	Mean Error	Std Dev	50% Bound	95% Bound	99% Bound
Light	RBS	$6.29\mu s$	$6.45\mu s$	$4.22\mu s$	$20.53\mu s$	$29.61\mu s$
Light	NTP	$51.18\mu s$	$53.30\mu s$	$42.52\mu s$	$131.20\mu s$	$313.64\mu s$
Heavy	RBS	$8.44\mu s$	$9.37\mu s$	$5.86\mu s$	$28.53\mu s$	$48.61\mu s$
Heavy	NTP	$1542.27\mu s$	$1192.53\mu s$	$1271.38\mu s$	$3888.79\mu s$	$5577.82\mu s$

reception interrupt before the packet is transferred from the NIC, which significantly reduces jitter and is a standard feature of the Linux kernel. The results in table 2.5 conclusively show the resilience of RBS against most error sources.

3 System Overview

Figure 3.1: Basic system overview

The basic system design is in accordance with the ARROWHEAD framework (fig. 3.1). In it's basic configuration the goal is to provide a distributed voltage measurement system. The system is split into 2 networks: a 6LoWPAN network consisting of several measurement nodes, and a network for the local ARROWHEAD cloud where the core services and the service consumer reside. The two networks are connected by a node which acts as a router and additionally offers clock synchronization in the 6LoWPAN network.

First the initial design decisions which led to component selections (both hard- and software) are briefly discussed, then the hardware platform (both commercially available and custom built components) are described. After this introduction the system is described in detail, outlining use-cases for the complete system and giving details of the inner workings of each component.

3.1 Initial design decisions

Several major decisions had to be made initially to design the distributed system. Decisions on a hardware platform and operating systems are necessary before any implementation can take place. This section documents the reasoning for these initial decisions. The implementation itself was a circular process, with both measurements and experiments leading to increased familiarity with the platform and the discovery of limitations, which in turn influenced the structure of the software.

Hardware platform

There exists a multitude of hardware platforms for embedded systems. Due to different wireless standards (e.g. Zigbee, 6LoWPAN, WLAN), operating systems (minimal ones like TinyOS, to IoT optimized systems like RIOT and CONTIKI, to full LINUX and Windows Embedded), processing power, etc. a great range of possible hardware platforms was available.

One platform that stood out was the MULLE platform [JVE+04] v2, which was developed by Eistec, a business spinoff of the Luleå University of Technology, which is very active in the ARROWHEAD project. Due to this connection this was the only platform which provided software support for the ARROWHEAD framework at the beginning of this thesis, with the supporting functions expected to grow as the ARROWHEAD framework develops. As usage of the ARROWHEAD framework was a constraint on the measurement system, this platform was chosen. For details on the MULLE see section 3.2.1.

To connect the 6LoWPAN network to external networks a Raspberry Pi 3 was chosen. This Linux based single board computer includes interfaces which can connect to the MULLE, to wired and to wireless LAN networks. As this component was periphery to the whole system, familiarity of the author with Raspberry Pi computers was the deciding factor to reduce development time.

Operating system

The MULLE platform supports two operating systems:

CONTIKI CONTIKI [DGV04] is an operating system for low-power wireless devices, optimized for Internet of Things devices. It also offers a network simulator called COOJA.

RIOT RIOT [BHG+13] is based on a microkernel architecture, optimized for a tiny footprint and real time abilities. It allows full C and C++ application programs and provides multithreading capabilities.

While the MULLE platform is an official platform for the RIOT operating system, most of the available software environment is only available for CONTIKI. As the MULLE platform was chosen to make use of the available software environment the RIOT operating system was no valid choice, even if its design methodology of a real-time system through and through would make it slightly better suited for synchronized measurement systems.

Network layer protocol

The CONTIKI operating system supports two main network layer protocols:

RIME RIME is a lightweight communication stack for wireless mesh networks [DÖH07].

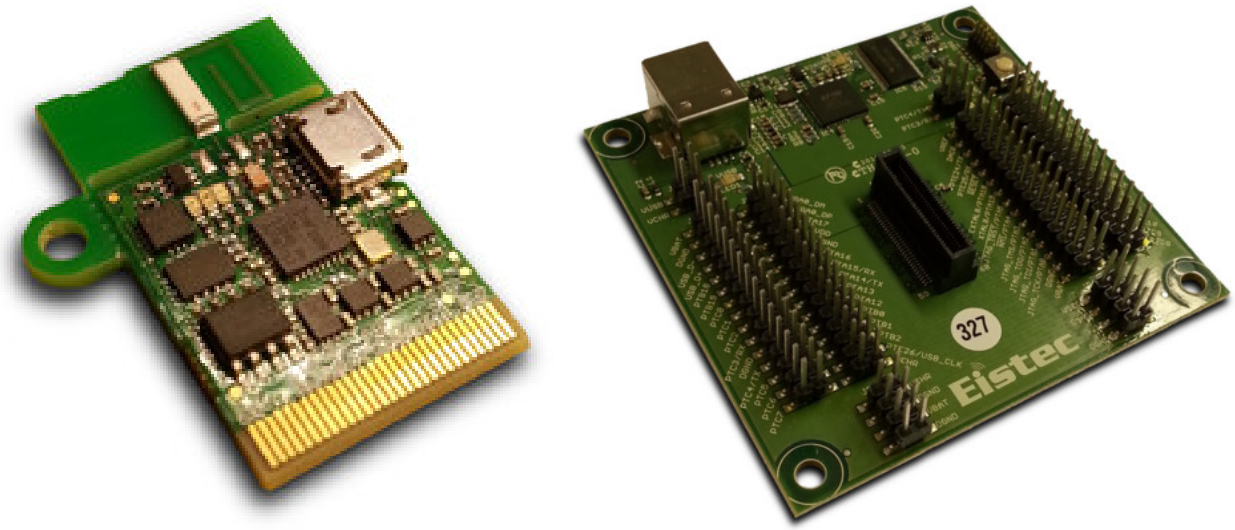
6LoWPAN IPv6 over low power wireless personal area network [MKHC07], which is a modified IPv6 implementation supporting header compression, fragmentation and routing in mesh networks.

For a local sensor network only RIME would be a very efficient protocol, however due to the service oriented nature of the ARROWHEAD framework global routing needs to be possible. To accomplish this a lot of overhead, e.g. IPv6 encapsulation on RIME, would be necessary, negating the benefits the protocol offers. Therefore 6LoWPAN, implemented in the uIP stack, was chosen as the main network protocol.
This decision however sacrificed the available time synchronization implementation for RIME [CON12]. However, as this synchronization was based on single direction clock updates, implemented by appending timestamps to all outgoing packets, its accuracy was too limited for the sub-ms synchronization needed.

Synchronization protocol

Under the assumption that all nodes are within communication range of each other, which can be assumed in the proposed measurement application at AVL List, the general purpose clock synchronization protocol PTP was chosen because it offered high synchronization accuracy, portability and relative ease of implementation. As the underlying principle of most mesh clock synchronization protocols (excepting the ones using special properties of the transmission like beacons) is the round-trip delay measurement, any knowledge gained using this general principle is applicable to other generic protocols.

The point of timestamping was chosen to be as close to the hardware as possible, without directly reprogramming the hardware itself. The point of using off-the-shelf hardware would have been lost if the radio driver would need to be completely rewritten. This would also make the implementation completely hardware dependent. Therefore it was decided to timestamp at the interface between the radio driver and the uIP network protocol stack, allowing for portability to any system using the CONTIKI operating system.



(a) MULLE **(b)** MULLE programming board

Figure 3.2: MULLE platform and programming board

Application protocol stack

Due to the developing nature of ARROWHEAD, only a partial environment was available while this work was being developed. However the ARROWHEAD framework can be used to tie different technologies together, managing the system. Service Registration and Discovery according to ARROWHEAD was implemented, however the applications it connected were using the OMA-LWM2M protocol. To limit resource use on the MULLE nodes a COAP to DNS-SD bridge was installed on the router, which meant that apart from PTP traffic all data packets were using the COAP stack.

3.2 Hardware

As the hardware platform influences many aspects of the project, a quick overview of the MULLE platform is presented below. The extensibility of the MULLE platform was further used to create an interface board for voltage measurements as its ADCs were not designed for $230V_{eff}$ input voltages.

3.2.1 MULLE platform

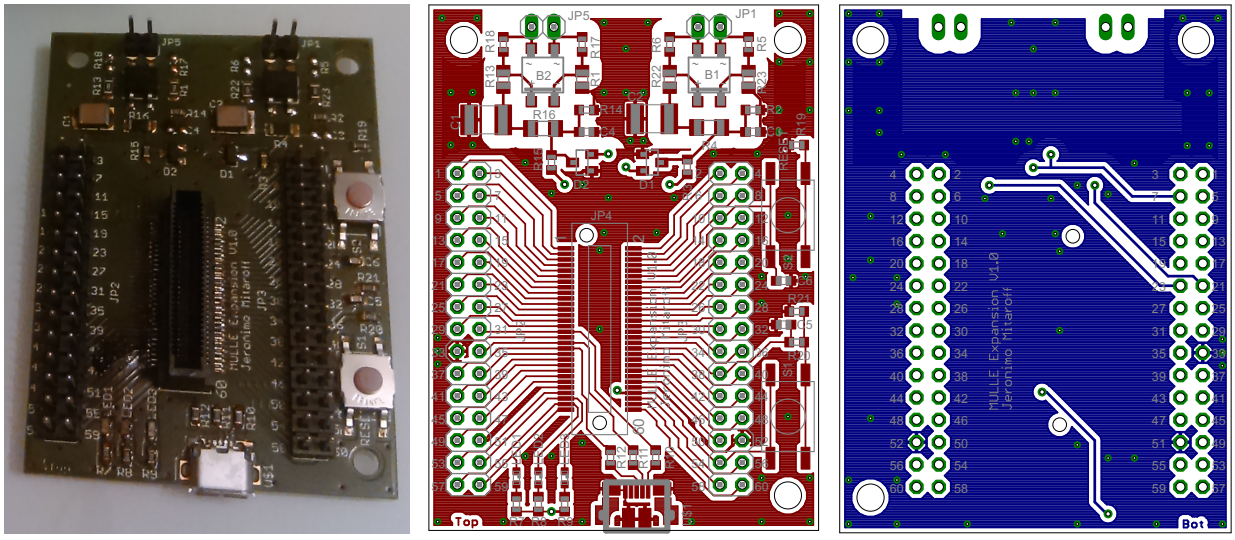
The MULLE platform [JVE+04] (fig. 3.2a) is an miniature embedded internet system. It includes a microcontroller fast enough to run a web server, 6LoWPAN communication, several sensors and a power supply. A battery is not included by default, however it can be easily connected. A board edge connector allows easy access to its interface. Due to its very small form factor it eases refitting, as the whole platform may even be placed within an existing enclosure.

The MULLE platforms main features are:

- ARM Cortex-M4 $100MHz$ microcontroller with $512KByte$ flash and $128KByte$ RAM
- $2MByte$ flash memory
- IEEE 802.15.4 $868MHz$ transceiver using 6LoWPAN or RIME
- microUSB connector
- LIS3DH 3-axis accelerometer
- Expansion header with 4x UART, 2x I^2C, 2x SPI, 2x CAN, USB-OTG, 18x ADC, 1x DAC and 42x GPIO pins
- On-Board voltage regulator
- ABS06 RTC quartz with $10ppm$ accuracy
- Compact design of $20.5x34x5.5mm$
- Support for multiple IoT technologies, including ARROWHEAD.

It comes with an optional programmer board (fig. 3.2b), which is necessary unless an USB bootloader is installed. The programmer board also offers a full pinout of the extension header and allows debugging via USB or JTAG.

3.2.2 Extension board



(a) Foto **(b)** Top **(c)** Bot

Figure 3.3: MULLE extension board after assembly (a), Layout top (b) and bottom (c)

To allow $230V_{eff}$ voltage measurements an extension board for the MULLE was designed and produced, as the programmer board would not withstand such voltages without external voltage level adjustments. As the programmer board additionally proved too expensive to provide for multiple nodes a full extension board instead of a plug-in board was designed. The extension board provided the following features:

- 1 MULLE extension connector, to plug in the small MULLE board.
- 2 measurement inputs for $230V_{eff}$ inputs, rectified and scaled to a maximum of $2.2V$ for the MULLE ADC inputs. Optionally they can also act as low-pass filters, easing voltage maximum measurements.
- 1 reset button
- 1 user button to control applications
- 3 status LEDs in the colors red, orange and yellow
- 1 USB connector, which can be used for power supply
- Full pinout of the MULLE for unforeseen applications

3.3 System description

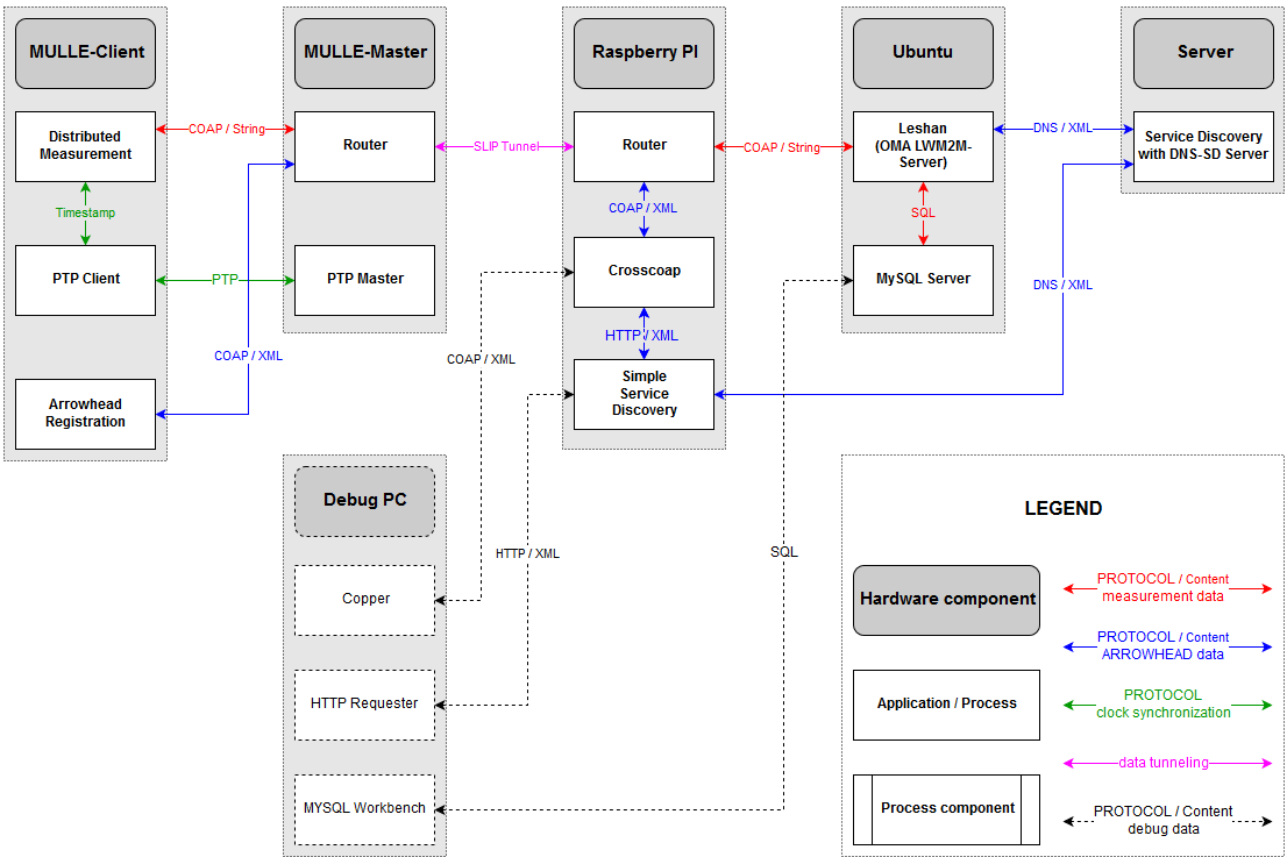


Figure 3.4: Overview of communication in the distributed test system

Figure 3.4 gives an overview of the different components communicating within the full test implementation. Each block represents a hardware component with different applications running on it. The arrows represent the communication path between these applications, detailing the protocols used for this communication. The components, from left to right, are as follows:

MULLE Client The MULLE platform in Client configuration. This includes ARROWHEAD registration, measurement and configuration resources and a PTP Client. The MULLE Client communicates via 6LoWPAN to the MULLE-Master. Up to 5 clients were tested simultaneously. See 3.3.1 for details.

MULLE Master The MULLE platform in Master configuration. This includes acting as a router, forwarding packets to the RaspberryPi with a wired SLIP connection, and acting as a PTP Master for the 6LoWPAN network. See 3.3.2 for details.

RaspberryPi The Raspberry Pi forms the second half of the routing pair, connecting the 6LoWPAN network to a classical 802.11 wireless network. Being advertised as a router it also provides an easily found COAP interface to the service discovery on the arrowhead server.

Ubuntu A PC running end user software, most notably a Leshan OMA-LWM2M server that is using the ARROWHEAD service discovery to find measurement nodes, connects to them via COAP and stores the measurement results in a MySQL database.

Server A PC running the ARROWHEAD CENTOS distribution, running a DNS-SD based service discovery server.

Debug PC During development several debug tools were used to verify the system over the network.

During operation, any nodes can register their services at the ARROWHEAD service discovery. The Leshan server periodically searches the service registry for MULLE measurement devices, and connects to them if new ones are found. The MULLE Clients, synchronized via PTP from the MULLE Master, provide their measurement data, which gets stored in a MySQL database for further use.

With this many interacting entities it is important to distinguish between parts which were available, and which had to be implemented during this thesis. The following notations are used in table 3.1:

Available The software/hardware was already available and had to be configured for the system.

Modified The software/hardware was already available, but had to be modified significantly. Check the appropriate chapter for details.

Created The software/hardware had to be created by the author of this thesis.

Table 3.1: Origin of entities

Name	Status	Chapter
	Hardware	
MULLE platform	Available	3.2.1
MULLE programmer	Available	3.2.1
MULLE extension	Created	3.2.2
	Software	
CONTIKI	Available	
PTP	Created	3.3.1, 3.3.2
Distributed measurement	Created	3.3.1
Border Router	Available	3.3.2
Tunslip	Available	3.3.3
Simple Service Discovery	Available/Modified	3.3.3
Leshan	Modified	3.3.4
MySQL	Available	3.3.4
Service Discovery	Available	3.3.5

3.3.1 MULLE Client

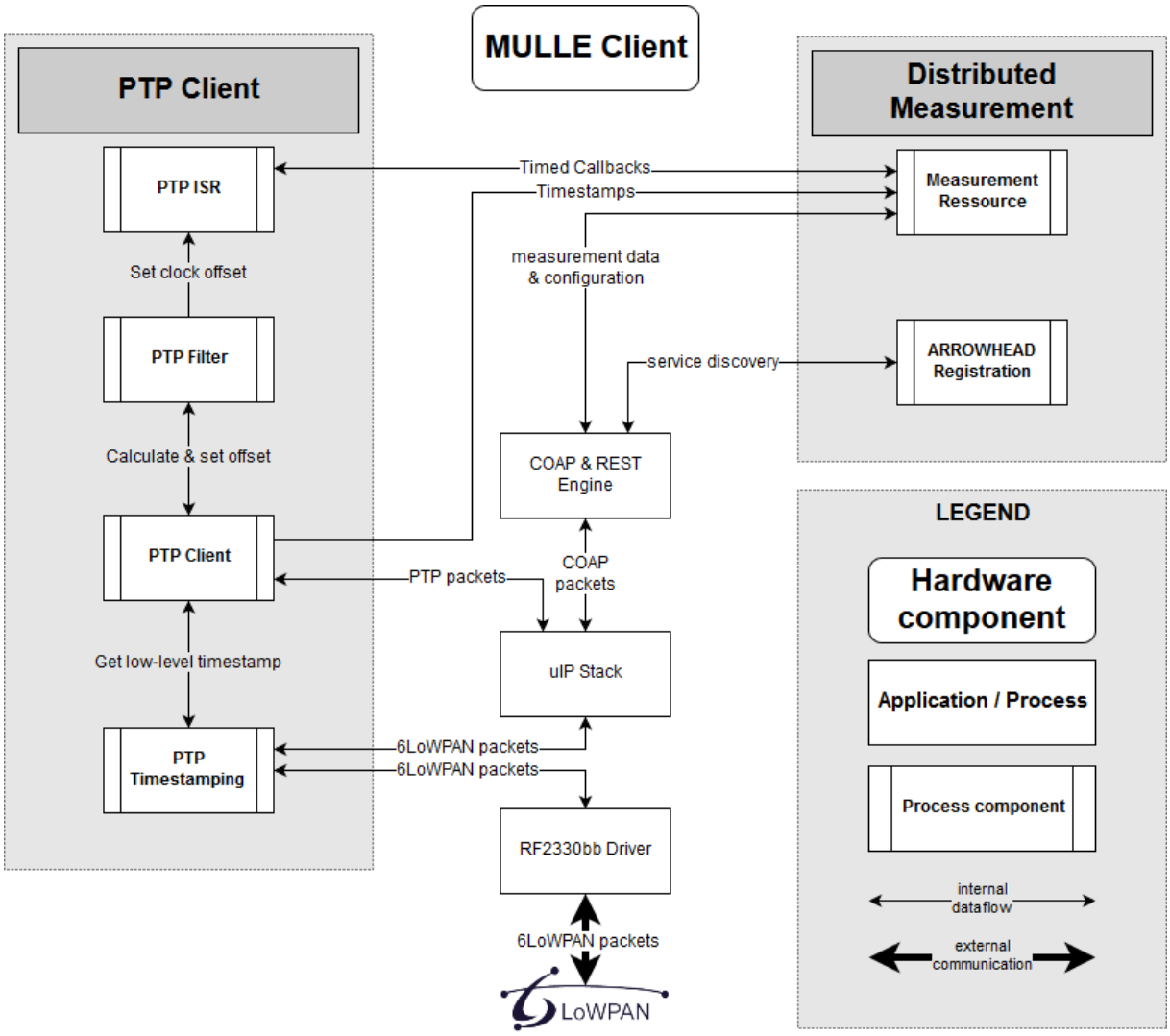


Figure 3.5: Internal dataflow of the MULLE client

The goal of the MULLE Client is to measure voltages across the network in a synchronized way, and offer these as a service. To do this several software components are needed, figure 3.5 gives an overview of the MULLE Client implementation. These are split into four running processes. Several system drivers are also shown in the center.

PTP Client The Precision Time Protocol Client implementation handles the synchronization of the internal RTC clock with the PTP Master. It is also connected on a very low level to the radio chip driver to timestamp in- and outgoing PTP packets.

Distributed measurement The measurement application for distributed synchronized voltage measurement across the network.

COAP & REST engine	Decodes COAP messages and handles RESTful resources. In addition to the custom measurement resources it manages multiple standard RESTful resources and server registration used for the OMA-LWM2M communication.
uIP stack	The small IPv6 implementation available on CONTIKI
RF2330bb driver	The low-level hardware driver of the radio transceiver on the MULLE platform.
Not pictured	Network initialization and router discovery process.

PTP Client

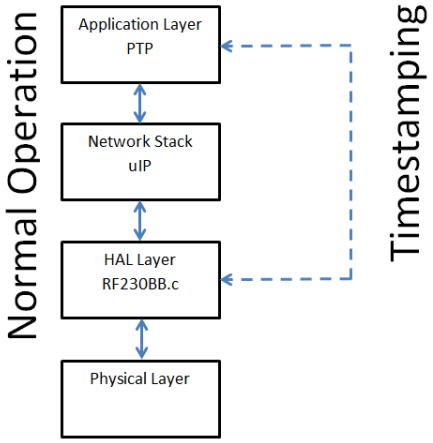


Figure 3.6: PTP timestamping

Because the uIP communication stack implemented in CONTIKI does not offer timestamping support, and also has a non-deterministic delay built into it, timestamping packets had to circumvent the hierarchical structure to occur on the lowest level possible, the RF2330bb radio driver (fig. 3.6). As the MAC layer was implemented in the RF2330bb chip itself, this meant timestamping happens at the interface between the MAC layer and the uIP network layer. First, all packets are timestamped to provide a single common point before any branching occurs. Packets of an appropriate length ($120 Bytes$ for single-cast, $76 Bytes$ for multicast) are checked if they are PTP packets. Then the PTP-packet number and the packet timestamp are saved in a ringbuffer. Afterwards the packet is passed on to the uIP stack for received packets, or to the RF2330bb radio for outgoing packets. If the PTP Client subprocess at the application layer later determines that the timestamp information is necessary it can retrieve it with the PTP-packet number from the ringbuffer. This way the timestamping algorithm is almost hardware platform independent, only a few lines have to be inserted at the interface between the radio driver and the uIP stack.

The handling of the PTP messages is split into four distinct parts (see fig. 3.5):

PTP Timestamping The low-level timestamping process, storing tags in a ringbuffer

PTP Client The message handling process, processes received PTP packets and periodically transmits DELAY-REQUEST packets. By itself PTP Client sends out DELAY-REQUEST unicast packets to the PTP Master every 4 seconds (PTP-standard). While SYNC and DELAY-RESPONSE packets are processed immediately, the clock offset of the internal RTC clock to the PTP Master is calculated at every FOLLOWUP packet. During this calculation the data is filtered to deal with jitter, interference, timestamping errors and RTC clock drift.

PTP Filter Applies filter algorithms to the clock synchronization, removing jitter and measurement errors. Several filter algorithms were implemented and evaluated, they are discussed in section 5.3.

PTP ISR This background task gets synchronized with each PTP packet. To allow external evaluation of the synchronization, a square voltage signal can be activated. The toggling of this signal gets shifted according to the measured time difference between the PTP Master and the PTP Slave, allowing to compare the synchronized signal with one generated by the PTP Master by measuring it with an oscilloscope. Alternatively, external tasks can register themselves to be triggered at specific points in time.

Distributed Measurement

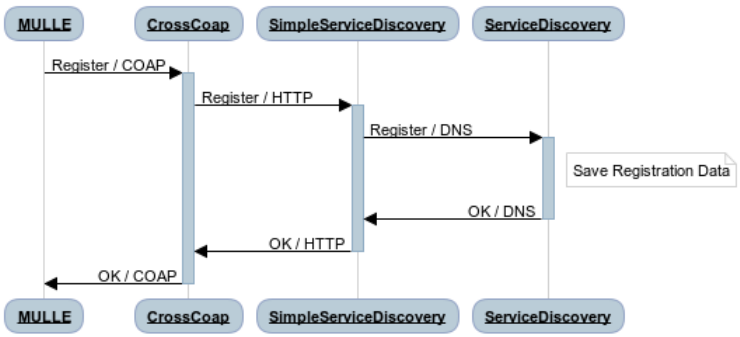


Figure 3.7: Flowchart of the Registration of MULLE

The Distributed Measurement ties all the parts together for a measurement solution. It handles the registration of the MULLE Client at the ARROWHEAD Service Discovery, sending its basic properties (name, IP address, port, type) XML-Encoded as a COAP POST to the CROSSCOAP interface. In addition it provides 2 dedicated resources:

- A configuration resource, which triggers the connection of the REST engine to an OMA-LWM2M server if a server IP address is received.

- A measurement resource, which measures the ADC voltage in defined bursts and stores them in a ringbuffer. A burst measurement consists of 16 voltage measurements, equally distributed within a $21ms$ window. The measurement bursts start time and interval between bursts can be configured remotely by a configuration message, and the measured data can either be read by COAP GET packets or simply OBSERVED for regular updates.

3.3.2 MULLE Master

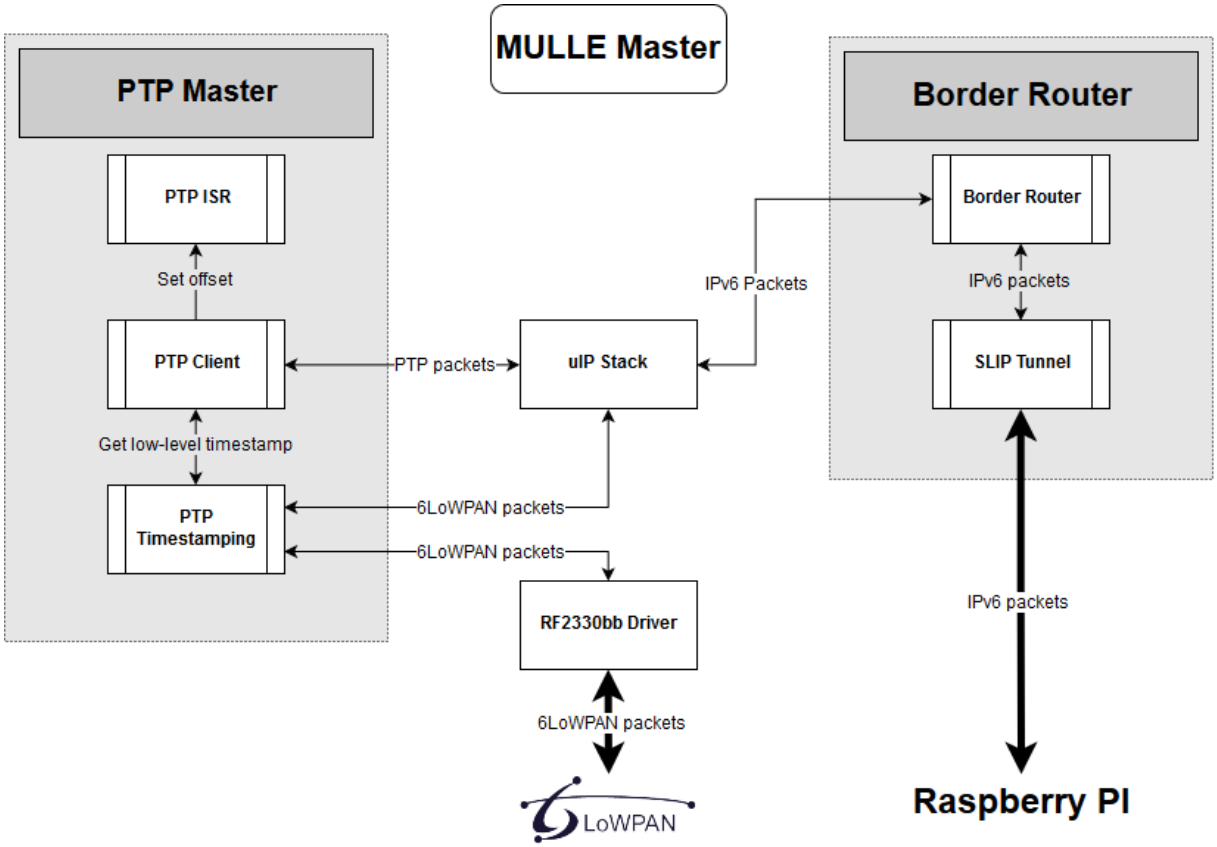


Figure 3.8: Internal dataflow of the MULLE Master

The MULLE Master was implemented very similar to the MULLE Client configuration. However, as the performance of the router affects the whole measurement system, no non-critical tasks were implemented on the MULLE Master. This means that only the PTP Master and the border router processes are active.

PTP Master

As the PTP Master only differs from the PTP Client (section 3.3.1) in its configuration, the system is implemented in the same way (fig. 3.8). However, as its RTC is the reference clock for the network no filtering of PTP data is necessary. The PTP Master regularly sends out SYNC and FOLLOW-UP packets to the local multicast group. A synchronization interval of 1.1 seconds was

chosen instead of the standard 1s PTP interval because it might encounter the same interference from other processes using the network, as multiples or fractions of a second are very common for a lot of processes. Offsetting the interval slightly allows the interference to be filtered out over multiple SYNC intervals.

If the PTP Master receives a DELAY-REQUEST packet, it immediately replies with a DELAY-RESPONSE packet including the receiving timestamp of the DELAY-REQUEST packet.

A background interrupt task which generates a square voltage signal has been implemented. This signal is tied to the PTP Master internal clock and used for external evaluation of the synchronization.

Border router

The border router process connects the 6LoWPAN measurement network to external networks, allowing global routing to and from the network. Any packets needing to cross boundaries get appropriate headers for their destination (6LoWPAN or IPv6) and are forwarded to the respective interface.

The MULLE Master is connected to the Raspberry PI via an UART connection using the SLIP (Serial Line IP) protocol [Rom88]. This protocol encapsulates IP packets for serial lines, and uses very little overhead. Therefore it is still widely used on microcontrollers. This allows routing any packets to and from an external network using the CONTIKI border router process.

3.3.3 Raspberry Pi Router

The main function of the Raspberry Pi is to forward packets to and from the 6LoWPAN network. Its full Ubuntu implementation however made it a convenient point to provide translation services which would have used extra resources on the CONTIKI operating system and to collect debugging data via Wireshark [Com16] as all packets going through can easily be examined.

Router

By using the Tunslip tool from CONTIKI a virtual network adapter using the SLIP protocol is created. This completes the connection by the MULLE Master border router process (3.3.2). By changing the network configuration in Raspbian, the Linux version running on the Raspberry Pi, routing between the SLIP interface and the external Ethernet network is activated.

Simple Service Discovery

The Simple Service Discovery [Mon16], provided by the ARROWHEAD framework, is a JAVA program that offers a bridge between COAP or HTTP packets and the DNS-SD service discovery

system. Requests are on ports 8045 (HTTP) or 5680 (COAP), and responses from the service discovery are sent back to the originating IP. This allows small hosts like microcontrollers to save on program code by just using a single protocol stack. The Simple Service Discovery was placed on the Raspberry Pi as all traffic to the service discovery is routed through this node anyways, and as the router the IP is automatically known by all other nodes.

However, lack of documentation of the required COAP message structure prevented the use of this part. As the JAVA program crashed immediately if a malformed COAP packet arrived. With no source code available and the possibility for malformed packets due to transmission errors on the wireless 6LoWPAN network it was decided to only use the stable HTTP interface. However, to keep the benefit of simplifying the code on the microcontrollers, CROSSCOAP [IBM16] was used to provide a local COAP to HTTP bridge. While this implementation adds some extra delay to any connection as two protocol translations are happening, these packets are not time-critical and it greatly eases development on the microcontrollers.

3.3.4 Leshan Server

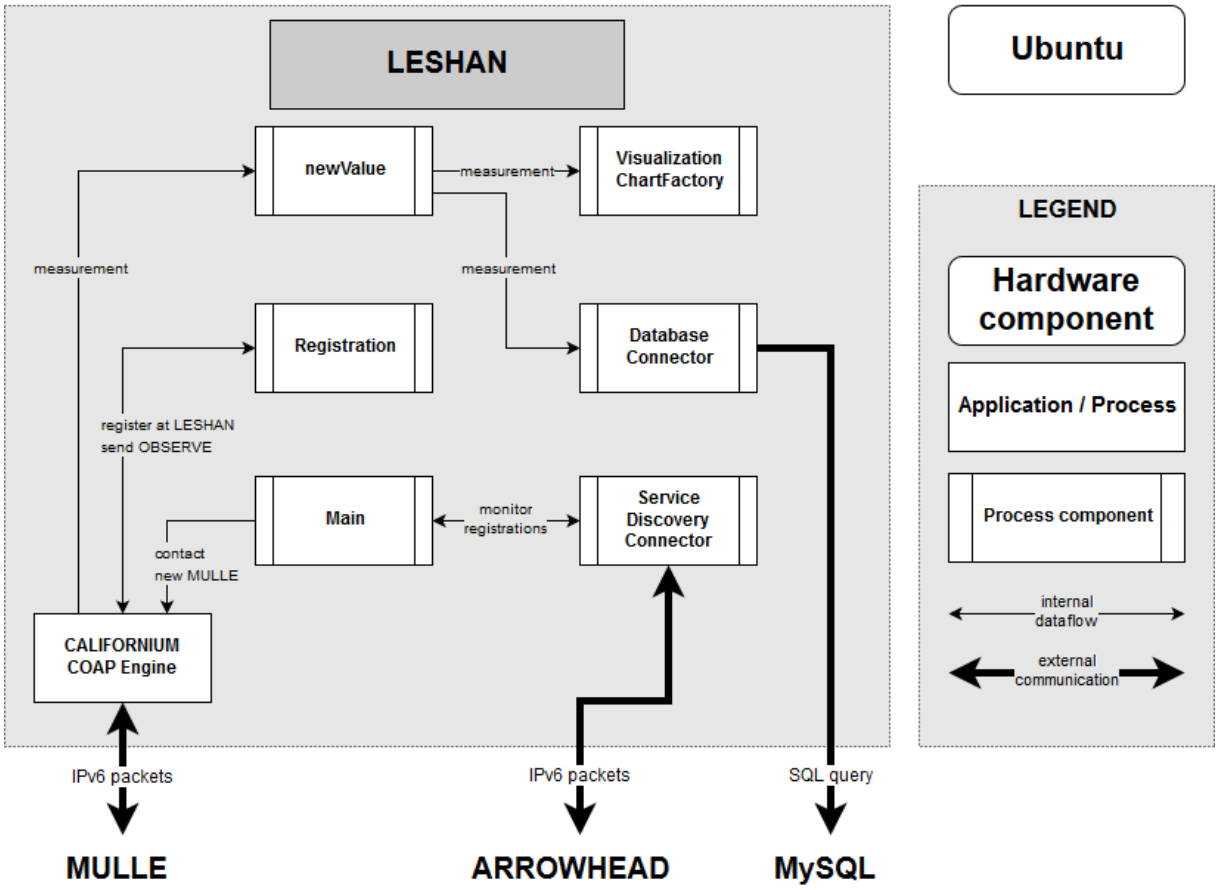


Figure 3.9: Internal dataflow of the LESHAN server

LESHAN [Ecl16] is a JAVA implementation of the OMA-LWM2M protocol, based on the CALIFORNIUM COAP implementation. Both servers and clients are available, and can be modified to suit the application. The LESHAN server acts both as a management server and a service consumer in the ARROWHEAD network. It's main tasks are:

- Maintain a list of active MULLE nodes. To do this the service discovery is scanned at periodic intervals for registered MULLE nodes.
- Connect to all active MULLE nodes to perform measurements. Observe appropriate resources to keep the overhead low.
- Save measurement data in a MySQL database.
- Maintain a clean service registry. This means to unregister any nodes which have stopped responding (e.g. due to power failure) after a certain timeout. The alternative, using a short time to live (TTL) period at the DNS-SD server to automatically remove the node there, was discarded. A TTL-solution at the DNS-SD server would need to be renewed before the entry expires, and for dynamic behavior of the system a short TTL is desired. This would create a lot of overhead. Because the LESHAN server regularly receives measurement data packets it can detect failure without any data overhead and within a short timeframe.

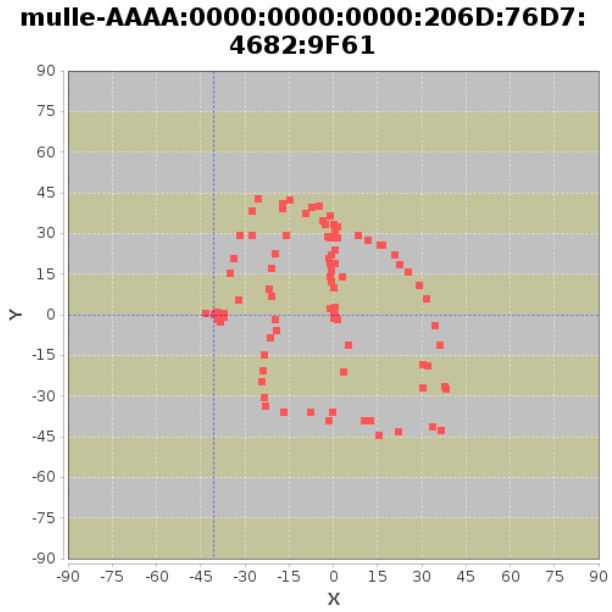


Figure 3.10: Tilt measurement of the MULLE, with history

Three different application scenarios were implemented:

Raw PTP data In this mode, raw PTP data is collected continuously and stored in the database. The collected data includes the timestamp differences of the SYNC and the DELAY-REQUEST messages (eq. 2.5 and 2.7) as well as the calculated clock offset between the PTP Client and the PTP Master. This made it possible to analyze raw PTP data in MATLAB, identify challenges (e.g. 4.3) and provide real data for filter development in MATLAB (section 5).

Voltage measurement During the synchronized voltage measurement each node sends the data gathered during a $20ms$ burst measurement each second. The data consists of 16 voltage measurements with their timestamps for each node. The Leshan server configures the nodes by selecting the measurement timestamp and frequency, which is distributed to all nodes. By calculating the zero crossing timestamp for each measuring node, the PTP synchronization accuracy can be estimated (section 6.1.3). Each measurement cycle will be displayed in a line plot (fig. 6.4) and saved in the MySQL database.

Tilt measurement For demo purposes and to test the service discovery, a tilt sensor application was also developed. In this mode each MULLE node measures its tilt by using the built-in accelerometer, and the data will be displayed in a separate XY-plot for each connected node (fig. 3.10).

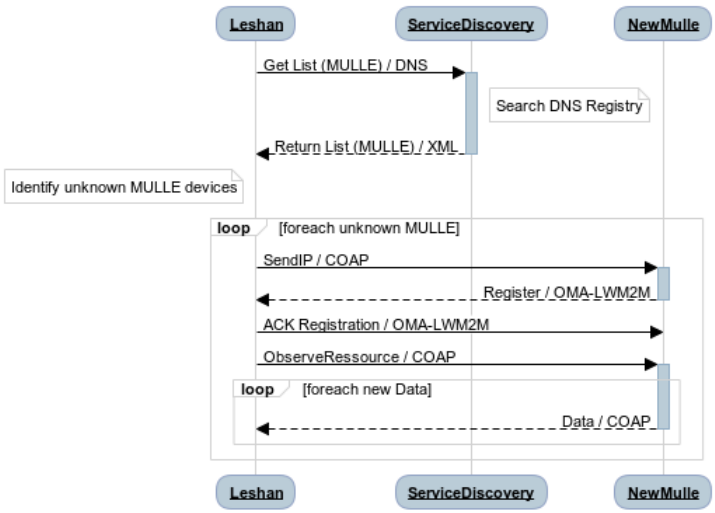


Figure 3.11: Flowchart of the Service Discovery

During start up, the Leshan server connects to the local MySQL database, and registers itself with the ARROWHEAD service discovery. Then it periodically requests a list of all registered MULLE Clients from the ARROWHEAD server. If a new client is found, the Leshan server sends a registration request to the MULLE Client. With the IP of the server now known, the MULLE Client registers at the Leshan Server. Once the registration is complete, the Leshan server requests to OBSERVE the appropriate MULLE measurement resource. The observed resource sends its data periodically to the Leshan server, which stores the data in a local MySQL database for future use.
Should a MULLE Client stop sending measurements, the Leshan server will remove it from the service registry after a short timeout.

3.3.5 ARROWHEAD Server

```
−<serviceList>
 −<service>
    <domain>0000.0000.0000.F95E.73B7.2C8A.43BA.</domain>
    <host>AAAA.0000.0000.0000.F95E.73B7.2C8A.43BA.</host>
   −<name>
      mulle-F95E73B72C8A43BA.mulle_coap._udp.srv.jmavl.ddns.net.
    </name>
    <port>5683</port>
   −<properties>
    −<property>
       <name>version</name>
       <value>1.0</value>
     </property>
    −<property>
       <name>path</name>
       <value>/mulle</value>
     </property>
    </properties>
    <type>mulle_coap._udp</type>
  </service>
</serviceList>
```

Figure 3.12: Registration of a MULLE node

The ARROWHEAD server consists of a CENTOS 6.4 Linux version, running a DNS server with service discovery extensions. By using the DNS-SD protocol, services can be registered, queried or removed from the local registry. Properties are organized using XML, and queries can filter by any properties. For each service the following top level tags are used:

domain The domain under which the service is available.

host The host name or IP address of the service.

name The name of the service. May be generated from different properties.

port The TCP or UDP port.

type The type of the service. Also includes the basic protocol required like *_udp* or *_tcp*.

properties A service might use additional properties to properly define itself. Here an arbitrary number of $<$ *property* $>$ tags may be listed.

As this server was also accessible from the rest of the internet under *jmavl.ddns.net*, the security configuration was tightened. The main improvement was a whitelist, only allowing local IP addresses to create DNS entrys and to start recursive querys.

4 Challenges of Software based clock synchronization

In this section challenges that only occur in software based time synchronization are discussed. First the problems will be described as it can affect all software based solutions, then additional details of how it affects the MULLE platform used will be given.

There are several areas where the synchronization uncertainty of a IEEE 1588 system can originate [FFR+11]:

- Uncertainty of the master clock
- Ability to select the best master clock
- Ability to distribute the master clock among nodes
- Ability of slaves to follow the master clock

In the implemented system the router is set as the best master clock by default, and this clock is assumed to be accurate. While this assumption only holds true within limits, as the router does not perform any additional tasks its RTC is the most stable within the system. This is described in detail in 4.3.

The ability to distribute the master clock among nodes can be influenced at several areas, each contributing to the jitter of the timestamps[FFR+11]:

- The timestamping capability of the master
- The timestamping capability of the slave
- The timestamping capability of the infrastructure
- Variability and asymmetric behavior of frame propagation delay.

While the timestamping capability of the master works well, timestamping at a slave can be influenced by non-interruptable tasks 4.3. As this work limits itself to a single-hop environment, no transparent clocks are available and therefore no effects of infrastructure exist. However, due

to the shared medium there is a non-deterministic variability to the frame propagation delay that is dependent on the network usage that is analyzed in 4.4. In addition, due to 6LoWPAN header compression asymmetric behavior exists 4.2.

The ability of the slaves to follow the master clock can be estimated by the accuracy of the RTC quartz crystals. The ABS06 quartz used has an accuracy of $\pm 10ppm$, which of course effects both the PTP master clock and the PTP slave clock, resulting in a relative worst case accuracy of $\pm 20ppm$.

4.1 Influence of timestamping point

The accuracy of the packet timestamping process is crucial for the accuracy of the synchronization. Timestamping should happen as close to the actual transmission as possible, as this results in less processes that can interfere with the timestamping. However this includes a tradeoff as these timestamping processes are platform dependent and therefore work against portability.

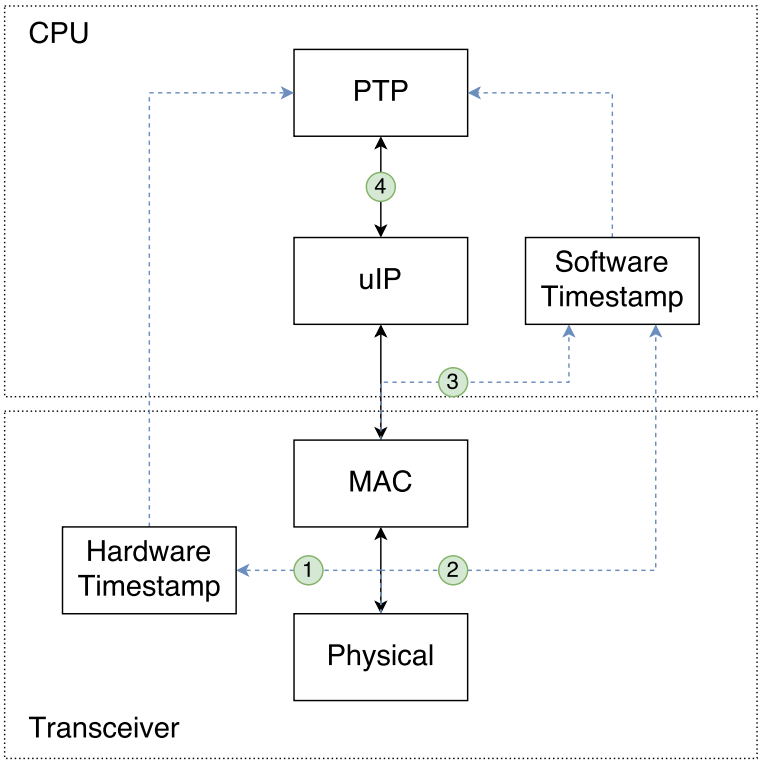


Figure 4.1: Timestamping points

The following timestamping points are available, beginning with the possibility closest to the transmission itself:

1. hardware Timestamping is done by specialized hardware in the transceiver at a defined point of the packet, both for sending and receiving

packets. This allows for accurate timestamping without any interference both from other software (4.3) or the network (4.4). Due to the defined timestamping point this also results in independence from asymmetric message length (4.2)

2. external interrupt — Some transceivers (including the RF2330bb used in the MULLE platform), while not having specialized timestamping hardware do have external interrupt lines which can be configured to trigger at specific points, e.g. "packet transmitted" and "packet received". These may be used to trigger external software timestamping in the CPU, therefore moving the timestamping point below the MAC layer included in the transceiver. This prevents the interference of the network (4.4), however interference from non-interruptable software (4.3) will still apply. This external interrupt line will be connected to the CPU in the next version of the MULLE platform.

3. software low level — Timestamping happens at the interface between the uIP stack and the MAC layer included in the external transceiver. This represents the most accurate option available with the current hardware, and is analyzed in detail in chapters 4 and 5.

4. software application layer — Timestamping happens at the application layer. This method is the easiest to port to different systems because the synchronization software is far removed from the hardware used, however in addition to the errors previously analyzed both the uIP stack and all other processes, not just non-interruptable tasks, will interfere with timestamping in a non-deterministic way.

4.2 Offset due to asymmetric packet compression

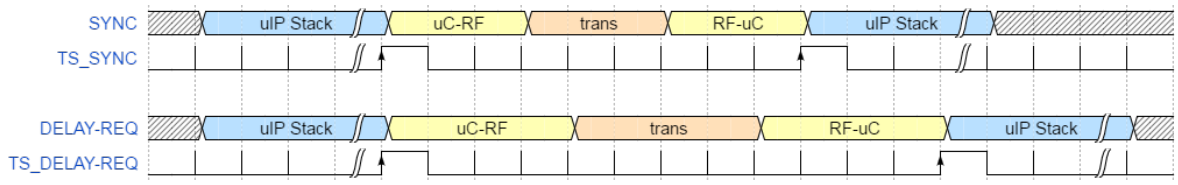


Figure 4.2: Effect of different packet lengths

The transmission delay compensation of PTP assumes that transmission delays are symmetrical between the PTP Master and the PTP Client. However, even with the same hardware, drivers and transmission path this does not have to hold true due to differing packet length. This discrepancy exists because SYNC packets from the PTP Master are sent out as multicast packets to the local multicast group FF02::0181, while the DELAY-REQUEST packets from the client are sent as unicast packets. Due to the header compression that 6LoWPAN uses (specified in RFC6282 [TH11]), the multicast header can be compressed better than an unicast. This is a result of the short number of relevant bits in the multicast group address.

Figure 4.2 shows the offset error when timestamping occurs right at the beginning of the transmission to the RF-chip, and right after a packet has been received from the RF chip. The bit rates for the connections CPU-radio and for the RF-link are assumed to be the same, the delay in the uIP stack is non-deterministic. The difference occurs in three places: first when the packet is transmitted from the CPU to the radio chip and buffered there, second when the packet is transmitted to the radio chip and buffered there and a third time when the data is transmitted from the radio chip to the receiving CPU. This offset increases the farther from the hardware the timestamping happens.

$$\Delta_{delay} = (length_{SYNC} - length_{DELAY-REQ}) * (\frac{1}{BW_{CPU-RF}} + \frac{1}{BW_{RF}} + \frac{1}{BW_{RF-CPU}}) \quad (4.1)$$

with

Δ_{delay}	The error in the delay measurement
$length_{SYNC}$	The length of the SYNC packet in bits
$length_{DELAY-REQ}$	The length of the DELAY-REQUEST packet in bits
BW_{CPU-RF}	The bandwidth from the CPU to the RF-Chip in bits per second
BW_{RF}	The bandwidth of the RF-channel in bits per second
BW_{RF-CPU}	The bandwidth from the RF-Chip to the CPU in bits per second

Because of the way PTP compensates for transmission delays (Equation 2.8) this results in a synchronization error of

$$\Delta_t = \frac{\Delta_{delay}}{2} = \frac{length_{SYNC} - length_{DELAY-REQ}}{2} * (\frac{1}{BW_{CPU-RF}} + \frac{1}{BW_{RF}} + \frac{1}{BW_{RF-CPU}}) \quad (4.2)$$

This error does not occur with hardware timestamping if both sender and receiver timestamp at the same place (beginning or end of the packet transmission) unless a buffering router or switch exists on the transmission path. A simple solution would be to force disable the header compression on all PTP-related packets, however this is not recommended as it would lead to increased overhead on the network. Because networks that use compression usually have limited bandwidth this could lead to increased conflicts. As this error is a static offset that can be calculated beforehand it is far more efficient to compensate the asymmetry in software.

Table 4.1: Communication delays on the MULLE

Communication Path	Speed	time SYNC $76Byte$	time DELAY-REQUEST $120Byte$	Difference $44Byte$
K60 CPU - RF2330bb radio	$6.076Mbps$	$100,1\mu s$	$158\mu s$	$57,93\mu s$
6LoWPAN radio	$100kbps$	$6080\mu s$	$9600\mu s$	$3520\mu s$
K60 CPU - RF2330bb radio	$6.076Mbps$	$100,1\mu s$	$158\mu s$	$57,93\mu s$
Total		$6280\mu s$	$9916\mu s$	$3636\mu s$

Specific to the MULLE Platform

On the MULLE platform this offset error could be measured. The header compression for unicast is turned off by default, while multicast packets are compressed. This results in a SYNC packet length of $76Bytes$, while the DELAY-REQUEST is sent as a unicast packet with $120Bytes$, leading to an asymmetry of $44Bytes$.

With the data rates the MULLE platform operates at (SPI-Connection $7.5Mbps$ with a $250ns$ delay after each Byte, effective $6.076Mbps$, $100kbps$ on the 6LoWPAN Connection) the resulting delay difference is $3,636ms$, resulting in a static error offset of $1.818ms$. A hardware interrupt might be possible from the RF2330bb, however the pin that is used for the timestamping interrupt is not connected in the current version of the MULLE platform. This change is planned by the hardware manufacturer in the third version.

During experiments (section 6.1.1) a static offset error of $59RTC - Ticks = 1.8ms$ has been measured, which is within 1 RTC-Clock LSB of the expected result. This value was obtained by having both the PTP Master and the PTP Slave output a synchronized $16Hz$ rectangle signal and measuring the average time difference between the nodes with an oscilloscope. Then a static asymmetry correction factor was introduced, which brought the signals into sync. The asymmetry had to be measured by external means as [FGK11] Theorem 1 proves that the asymmetry cannot be measured by pairwise round-trip measurements.

While this is a worst case (uncompressed unicast vs. compressed multicast), this confirms the correctness of the assumptions. The error is reduced by activating header compression for the whole CONTIKI system, but the fact remains that the FF02::0181 multicast address will always compress better than an unicast address. With a delay difference of $82,63\mu s$ per Byte any length difference remains relevant.

4.3 Interference from periodic, non-interruptable tasks

Periodic, non-interruptable tasks can present a problem for software clock synchronization. If a SYNC-packet is received during a phase where the processor cannot be interrupted, the packet can only be timestamped once the task ends. This adds a significant positive error to the timestamp,

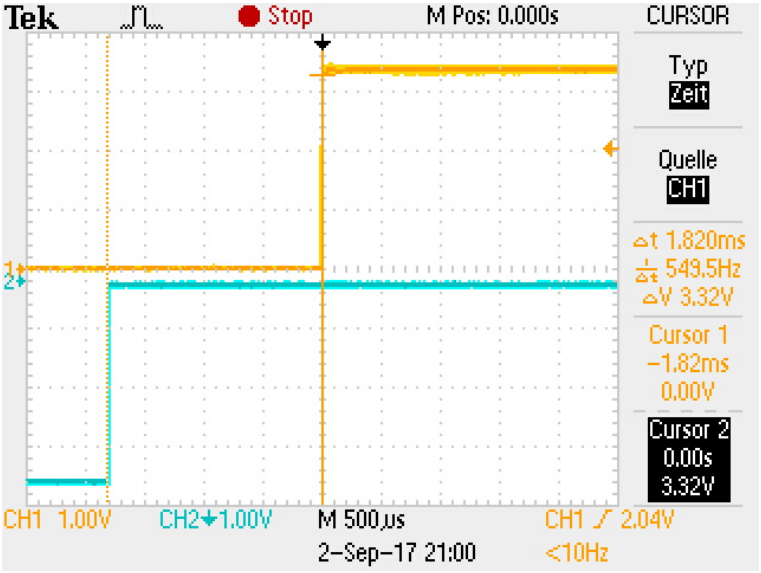


Figure 4.3: Measuring synchronization asymmetry with oscilloscope

and therefore to the synchronization. Fig. 4.4 illustrates this for a fixed-length non-interruptable task

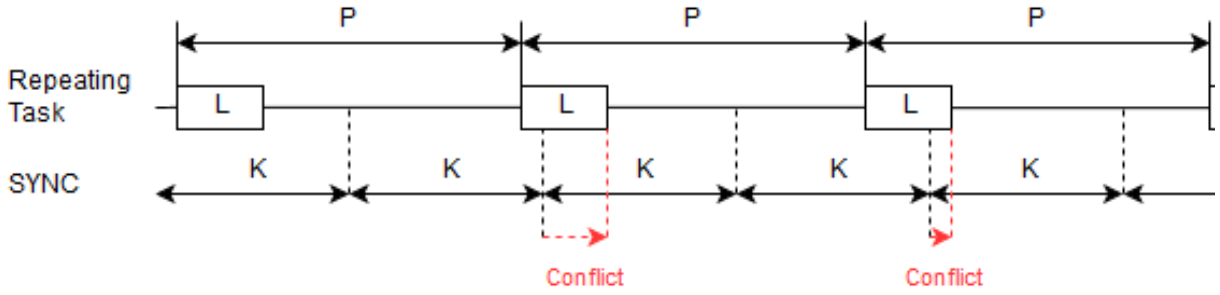


Figure 4.4: Non-interruptable task vs SYNC

with the period P and length L. The possible conflict with the SYNC-packets arriving with an interval of K is clearly visible. To mathematically describe this conflict the conditions

$$n, m \in N, \quad P, K > L \tag{4.3}$$

$$m * P \leq n * K < m * P + L \tag{4.4}$$

have to be defined. For simplicity of the equations a common origin time for both the non-interruptable task and the SYNC interval is assumed at $t_0 = 0$. This can be refactored into

$$0 \leq n * K - m * P < L \tag{4.5}$$

With these conditions it is possible to solve for

$$m = \lfloor n * \frac{K}{P} \rfloor \tag{4.6}$$

leading to a resulting synchronization packet timestamp

$$t_{SYNC}(n) = \begin{cases} \lfloor n * \frac{K}{P} \rfloor * P + L & \text{if } (n * K) \mod P < L \\ n * K & \text{otherwise} \end{cases} \tag{4.7}$$

.

This results in a periodic error pattern which can be seen in Fig. 4.5. Because of the discrete nature of the equation multiple error-patterns can overlap, with one conflict occurring at the beginning of L and another at the end. The values chosen for this simulation are $P = 10000, L = 1016, K = 35842$, which corresponds to the measurement of this error in section 4.3.

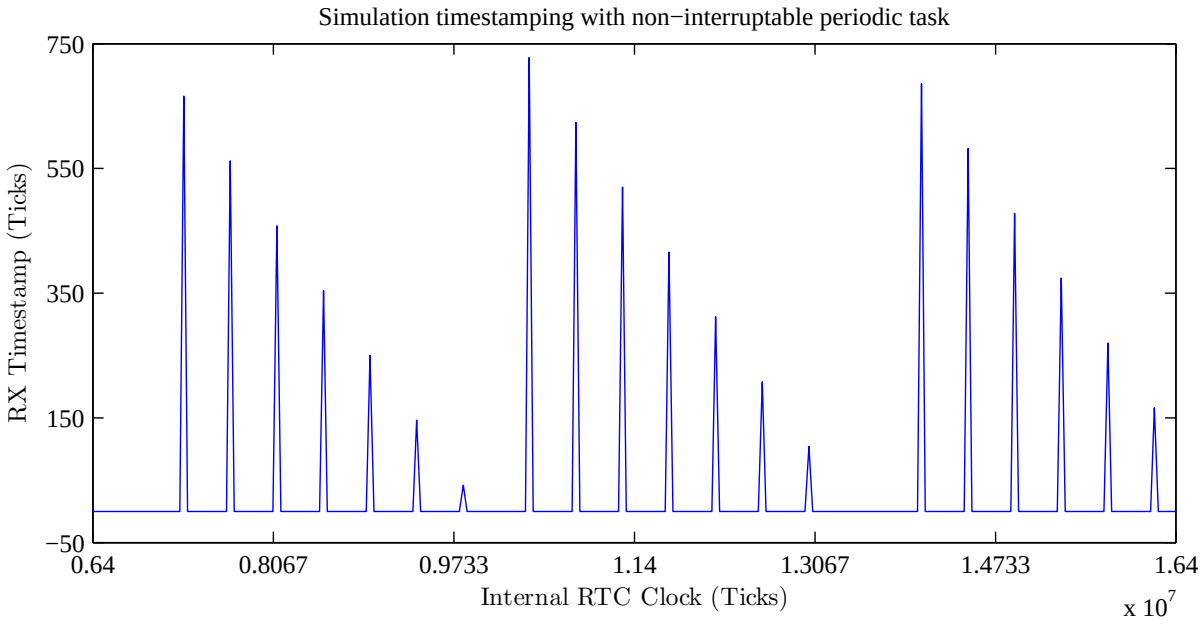


Figure 4.5: Non-interruptable task vs SYNC simulation

Specific to the MULLE Platform

A peculiarity of the MULLE platform is that the real time clock becomes unstable if it is used to generate interrupt timings where the last 9 Bits of the RTC are used. In this case the accuracy of the RTC clock to the reference clock in the PTP Master is reduced from $7ppm$ to 0.6%. This seems to be a side-effect of the RTC-implementation of CONTIKI, which uses a 512 tick grid [CON16] to avoid scheduling conflicts. This is especially problematic because this clock drift is unstable, depending on the number of interrupts called and their position. Therefore these states were avoided by only triggering interrupts on exact RTC multiples of 512 and delaying inside the interrupt until the desired clock position has been reached. However, this adds a dynamic component L_D to the length of the non-interruptable task (Fig. 4.6) before the desired execution point of $m * P$.

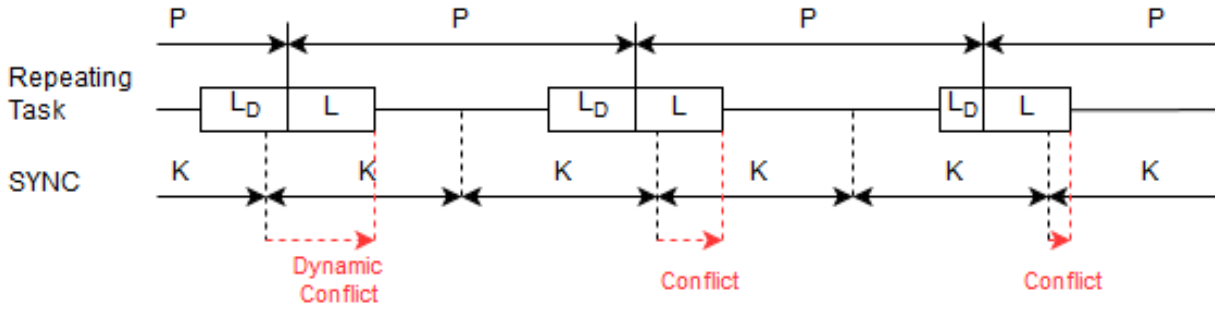


Figure 4.6: Non-interruptable task with dynamic component vs SYNC

To calculate the offset errors equation (4.7) can be expanded to include the new dynamic length of the interrupt. The new dynamic interference condition can be described by

$$n, m \in N, \quad P > L, \quad P > 512 \tag{4.8}$$

$$m * P - L_D \leq n * K < m * P \tag{4.9}$$

$$0 \leq m * P - n * K < L_D \tag{4.10}$$

$$L_D = (m * P) \mod 512 \tag{4.11}$$

Similar to the previous condition it is possible to solve

$$m = \lceil n * \frac{K}{P} \rceil \tag{4.12}$$

resulting in the interference condition

$$\lceil n * \frac{K}{P} \rceil * P - n * K < (\lceil n * \frac{K}{P} \rceil * P) \mod 512 \tag{4.13}$$

.

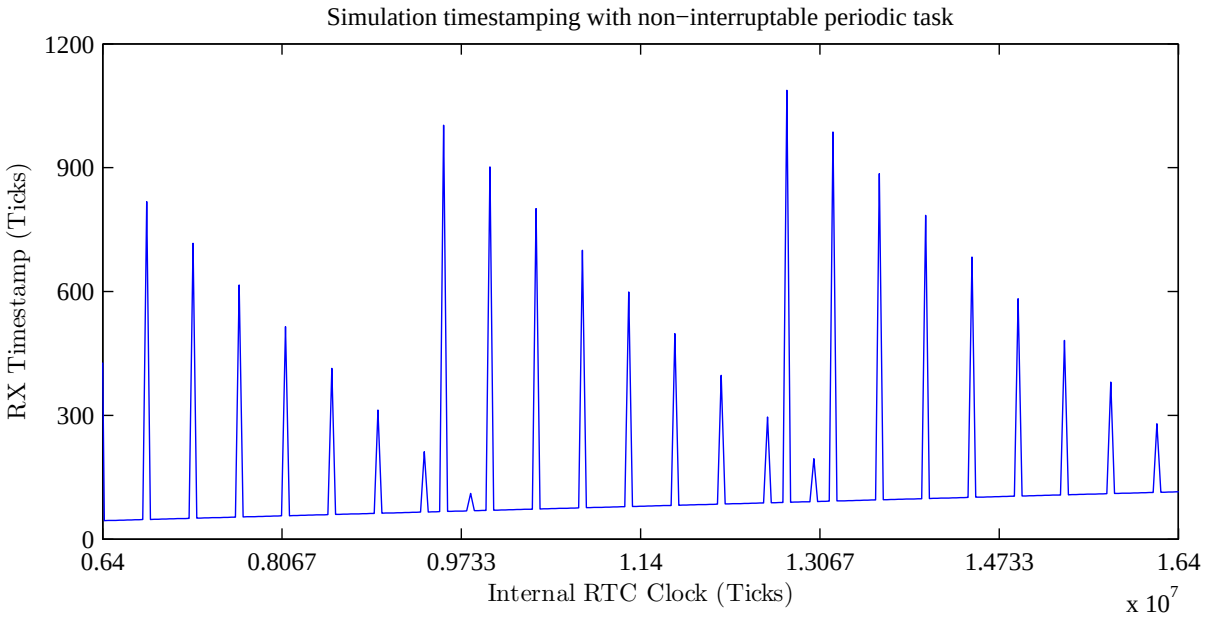
Expanding the results from the non-dynamic case (4.7), the measured timestamp can finally be described by

$$t_{SYNC}(n) = \begin{cases} \lfloor n * \frac{K}{P} \rfloor * P + L & \text{if } (n * K) \mod P < L \\ \lceil n * \frac{K}{P} \rceil * P + L & \text{if } \lceil n * \frac{K}{P} \rceil * P - n * K < (\lceil n * \frac{K}{P} \rceil * P) \mod 512 \\ n * K & \text{otherwise} \end{cases} \tag{4.14}$$

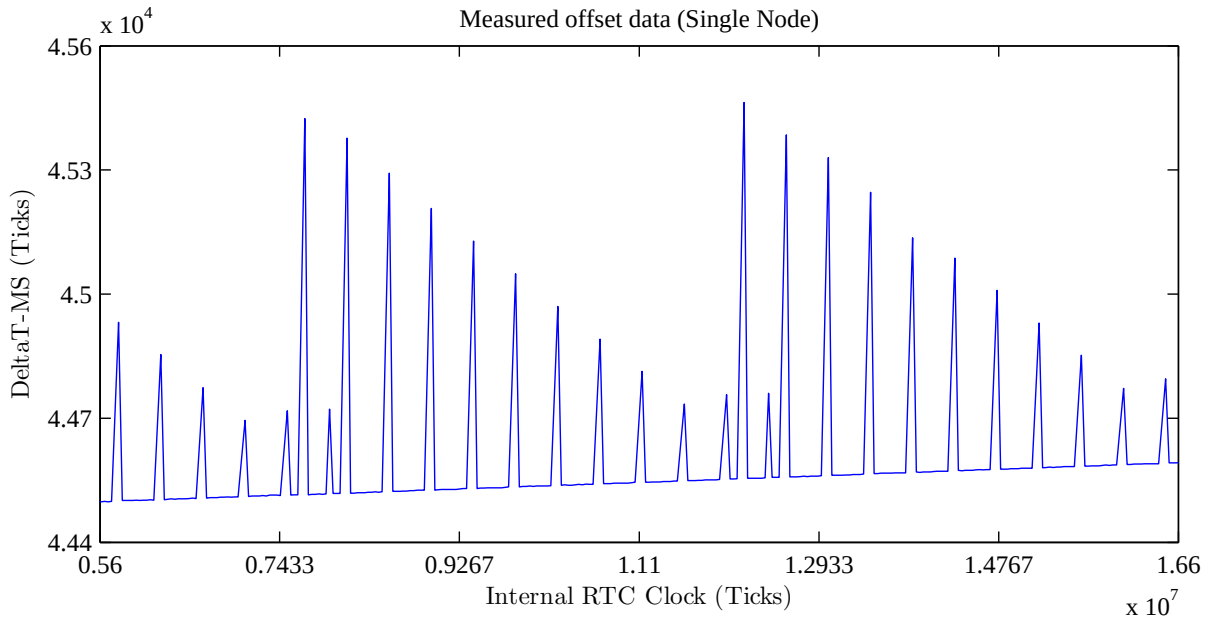
.

Eq. (4.14) combines (4.7) and (4.13) to fully describe the interference. While not well suited to analytical mathematics it allows numerical simulation, as can be seen in Fig. 4.7a, with $P = 10000, L = 1016, K = 35842$ and a clock drift of $7ppm$ added as well. This compares well to the measurements taken with the MULLE PTP Client (Fig. 4.7b) using a RTC frequency of $32768Hz$, with $P = \frac{10000}{32768}s = 305ms$ and $K_{SYNC} = \frac{35842}{32768}s = 1.1s$. The effect of the limited accuracy of the quartz oscillators, causing the local clocks to slowly drift apart, is clearly visible in the linear increase of the clock offset between the PTP Master and the PTP Slave.

This error is unlikely to occur with the DELAY-REQUEST message of the client because the interrupting task has to be triggered in the small window between timestamping and the transmit command, while it blocks received SYNC packets during its whole length. On the PTP Master such a task would however interfere heavily with this communication.



(a) Simulation including drift



(b) Offset Measurement

Figure 4.7: Simulated (a) and measured offset error (c) on MULLE due to non-interruptable tasks

4.4 Interference from network conflicts

As the system uses the IEEE 802.15.4 unslotted mode, nodes do not have specific timeslots to access the common medium. Therefore media access conflicts may occur. Because the timestamping is done before the packet enters the MAC layer, delays due to the shared medium cannot be automatically corrected. This section analyzes the impact of these conflicts on timestamping, section 5.3.6 will present filter improvements specifically to deal with the resulting timestamping

errors.

The shared medium uses CSMA/CA for arbitration, which is based on monitoring the medium for other transmissions and waiting until the medium is free. If conflicts occur, all parties employ a random exponential backoff algorithm to avoid deadlocks. The backoff algorithm for IEEE 802.15.4 uses a basic backoff period of 20 symbols, encoded with offset-quadrature phase shift keying of $4bits/symbol$, resulting in backoff periods that are multiples of $80bits$.

The simulated system is based on the following assumptions:

- One node is acting as a PTP Master, transmitting a SYNC packet every $1.1 + xs$, starting at $t = 0$. A random component x with a maximum of $21.4ms/700ticks$ has been added to the period to avoid continuous conflicts with other periodic transmissions, leading to an average period of $1.1107s$. The maximum of the random component has been chosen to be slightly larger than the longest packets to break persistent conflicts, but small relatively to the period to keep the simulation consistent. The SYNC packet is immediately followed by a FOLLOWUP packet.
- N nodes are acting as PTP Slaves, transmitting a DELAY-REQUEST packet every $4 + xs$, immediately followed by a DELAY-RESPONSE packet from the master, starting at a random point within the first $4s$. To avoid multiple nodes with a close starting time to continuously interfere with each other the same random period x has been added, leading to an average period of $4.0107s$.
- Each PTP Slave in addition sends a data packet once per second, this represents the measurement application data. Two variants of these data packet schedules are analyzed: one where the data is sent from a random starting point, and one where after synchronization has occurred virtual sending slots are imposed on these transmissions. Because a working synchronization is a prerequisite for this conflict avoidance mechanism, no such slotting system is assumed for the PTP packets themselves.

The conflict arbitration is modeled after [BV09] which conducts a performance analysis of IEEE 802.15. 4 non beacon-enabled mode. One of the results of their extensive network simulations is that the cumulative transmission probability is close to an exponential function

$$F_T = 1 - e^{-j/\tau_i} \tag{4.15}$$

with j representing the sending step and τ_i determined by the number of conflicting nodes, these functions can be used to approximate the arbitration while being efficiently simulated. Curves for up to 6 nodes in conflict at the same time were being generated (fig. 4.8). Each time a conflict happens during simulation a random probability is generated and the formula is reversed to calculate the corresponding successful sending step. For simulation efficiency, conflicts greater than 6 nodes are using the 6 conflicting node function. This however is a simplification. The goal of these simulations is to allow for error estimations, they do not simulate the full CSMA/CA process. In addition, data communication is simulated as distinct nodes, while in reality the slaves share their measurement data. This allows a node to conflict with itself, slightly increasing conflict chances. While conclusions can be drawn from these imperfect models, this limitation has to be explicitly stated. Especially at high duty cycles a full CSMA/CA simulation will return

different results, however this was outside the scope of this thesis.

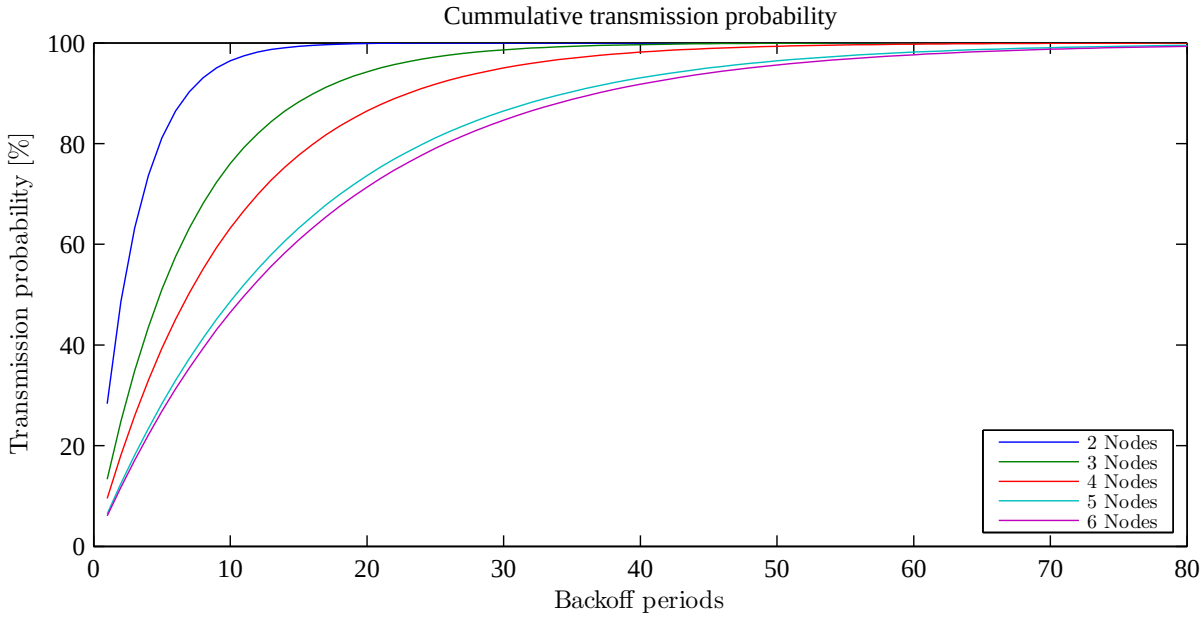


Figure 4.8: CSMA/CA backoff probabilities

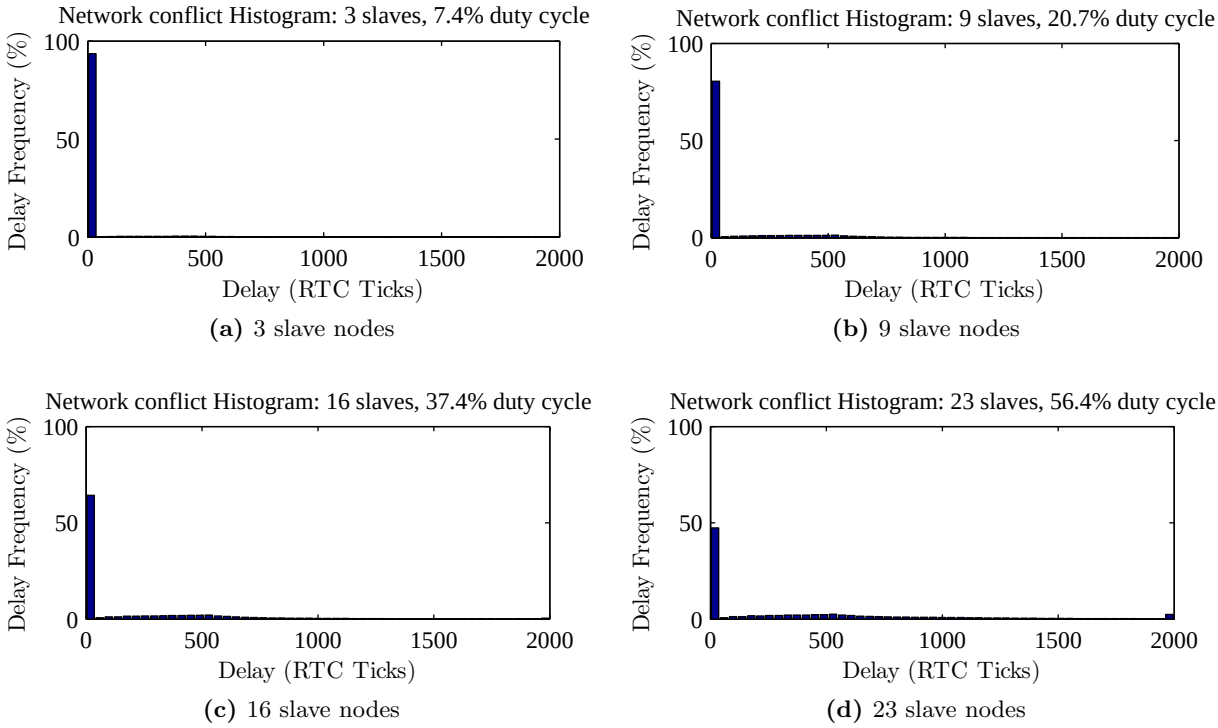
Table 4.2: Simulation Parameters

Bitrate	$100kbps$	
Time Resolution	$1/32768s$	
Unicast Length	$120Byte$	$315ticks$
Multicast Length	$76Byte$	$200ticks$
Data Packet Length	$200Byte$	$525ticks$

Simulations are being done with the network containing 1 to 23 slaves. To average the random conditions, the data from 350 simulations, each consisting of an hour of data, are being merged.

Unslotted Mode

Figure 4.9 analyzes the delay distribution incurred by the master and the slaves. As can be expected due to the uncorrelated manner of the messages, the zero delay bar can be estimated by the probability $p = 1 - dutycycle$, the chance that the channel is free. The slight discrepancy to the estimation stems from the fact that the lowest bar includes delays up to 50 ticks, which includes the shortest possible backoff period. With the limited amount of data in each simulation (slightly above 8 million packets at 23 slaves) the histogram becomes less readable if the bins get too small because a lot of fluctuation between neighboring bins exists. Due to the backoff algorithms used the delay distribution does not decrease monotonously, instead it experiences a slight maximum



(a) 3 slave nodes

(b) 9 slave nodes

(c) 16 slave nodes

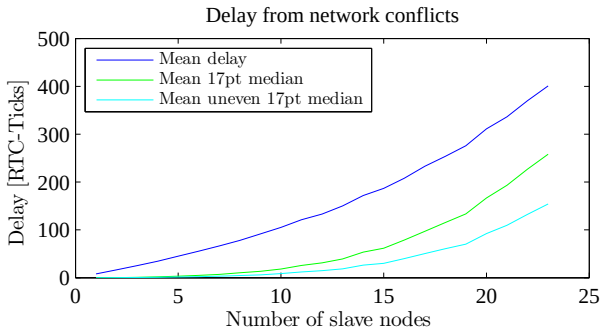
(d) 23 slave nodes

Figure 4.9: Unslotted mode delay distribution of network conflicts for 3 (a), 9 (b), 16 (c) and 23 (d) slaves

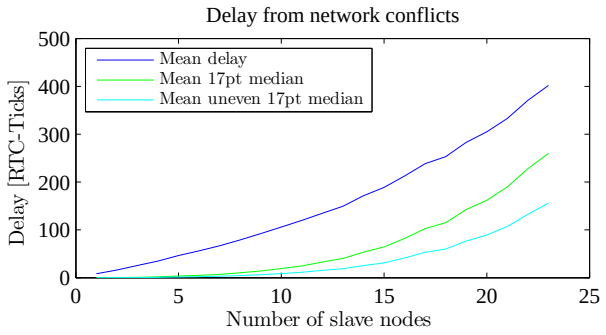
between 300 and 500 ticks. This is dependent on the average number of nodes colliding, as multiple conflicts take longer to resolve. The increase in the last delay bar of the 23 slave scenario is only due to the axis of the diagram, as all delays greater than 2500 are also included in this bar.

As can be seen in fig. 4.10d due to collisions the overhead (red line representing the overhead compared to the channel bandwidth, the green line represents the overhead relative to the useable data) increases in an exponential manner. This results in a rapid increase of both the average delay (fig. 4.12a) and the average deviation (4.12c, which can be delayed by using median filters (green and blue lines). As periods that do not share divisors were chosen and the packets are uncorrelated, the difference between randomized and non-randomized SYNC periods are minute (fig. 4.10b).

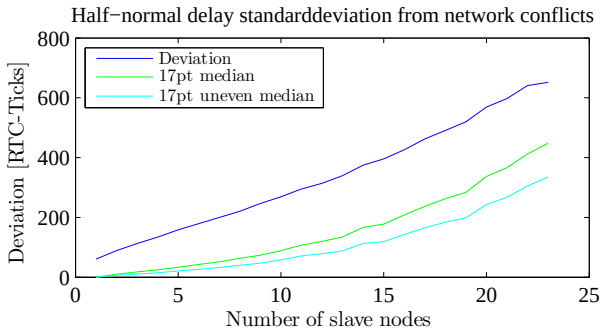
Once the median filter reaches a certain size the error patterns begin to repeat themselves. As this results in data based on error probabilities, increasing the length of the filter does little to affect the outcome. However it is possible to change the median algorithm to an uneven median filter. Figures 4.12e and 4.10f analyze the effect of moving the selection point. In section 5.3.4 this kind of filter is analyzed in further detail. However it should be no surprise that moving the selection point improves the filtering, as in this simulation only the current one-sided error model is included. In this scenario a minimum delay selection algorithm, as is used in NTP [MMBK10], would provide optimal results. However this would ignore other types of timestamping errors, therefore a lower limit on the selection is imposed.



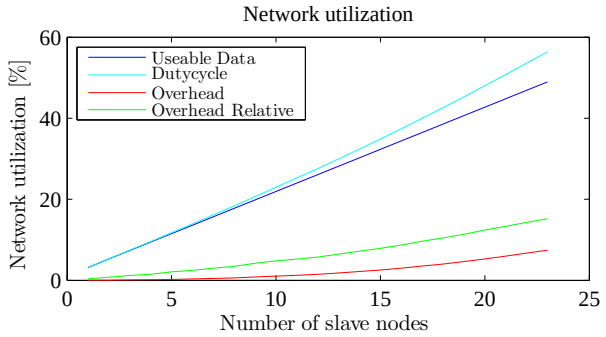
(a) Average delay of all timestamps



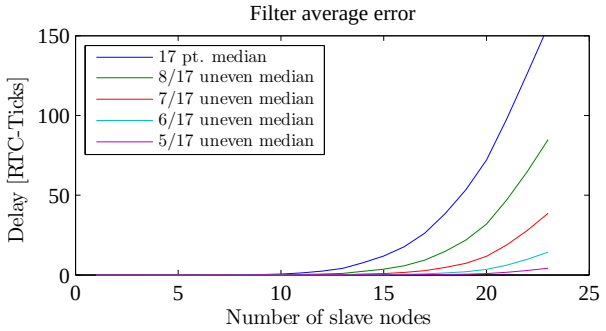
(b) Average delay, all timestamps, no random SYNC period



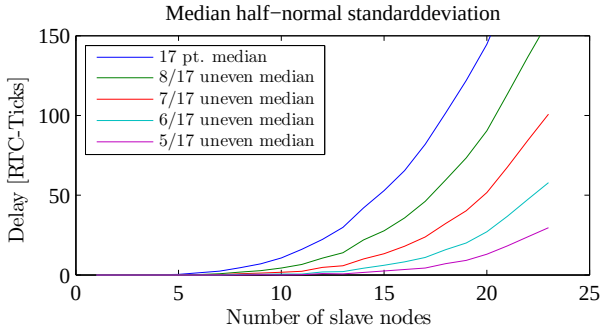
(c) Average deviation



(d) Duty Cycle



(e) Filtered SYNC delay

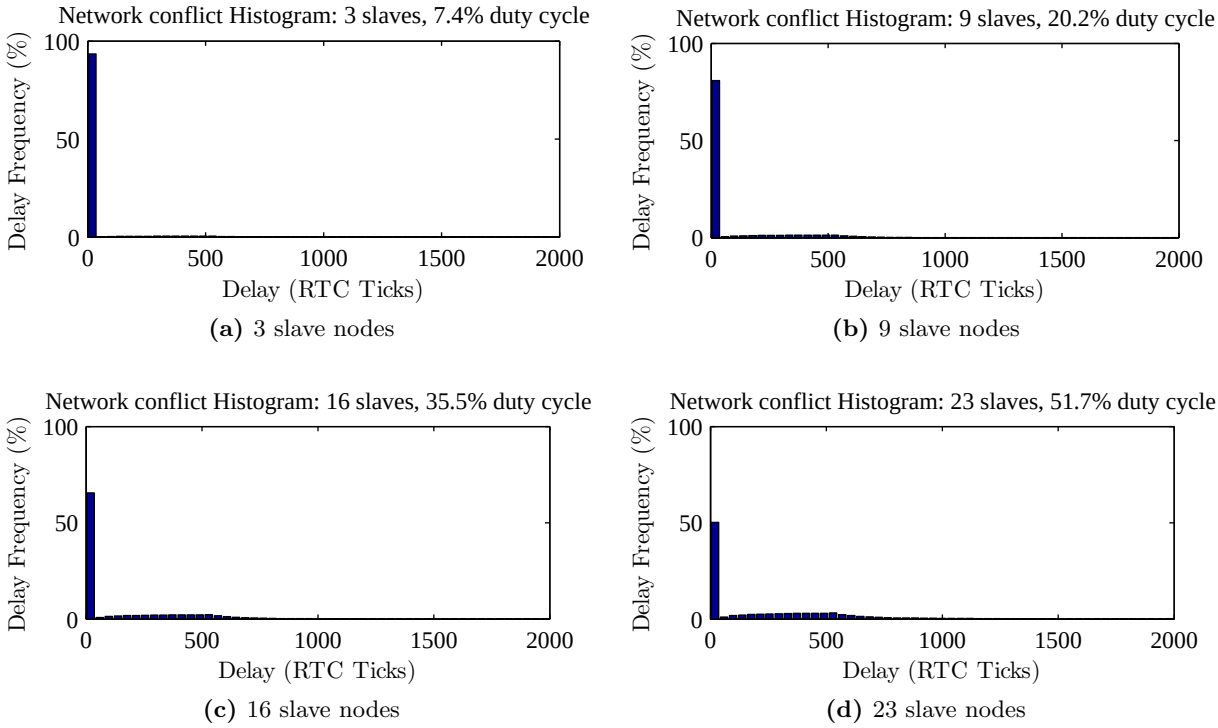


(f) Filtered SYNC delay deviation

Figure 4.10: Unslotted mode delay, standard deviation and duty cycle

Slotted Data Mode

In slotted data mode, the background data traffic is scheduled to be sent at regular intervals. In the simulation this is accomplished by only scheduling a single data source with a period of $\frac{1}{N}s$, where N represents the number of slaves. The delay increases almost linearly until 15 slaves are reached, after that arbitration delays increase exponentially. While faring a lot better than the unslotted mode, once multiple conflicts occur the delay increases sharply. In addition, even with the randomized periods, a slight resonance can be seen in 4.12a when analyzing the performance



(a) 3 slave nodes

(b) 9 slave nodes

(c) 16 slave nodes

(d) 23 slave nodes

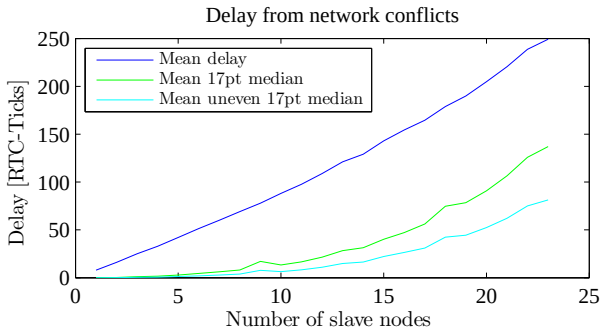
Figure 4.11: Slotted mode delay distribution of network conflicts for 3 (a), 9 (b), 16 (c) and 23 (d) slaves

of the median filter. At 9 slaves the data period $T = \frac{1}{9}s = 0.1111s$ gets close to the average SYNC period of $1.1107s$ with every tenth packet. The same happens at 18 slaves with double the data packets interfering. A slight resonance can also be detected at 14 slaves, where every 31^{st} data packet comes into conflict with every second SYNC packet. Because the data period changes depending on the number of slave nodes this cannot be completely avoided unless the SYNC period is dynamically adjusted at the same time.

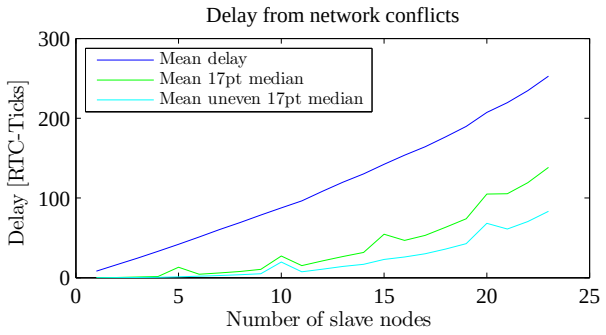
Comparing with the resonance occurring without the randomized SYNC period (fig. 4.12b), the improvement can clearly be seen. However, it is important not to lose sight of the SYNC path in the aggregated timestamping errors. Even with a lot less of the traffic and therefore timestamps, the SYNC path accounts for half of the synchronization process. Therefore it is important to analyze the SYNC interference separately from the rest of the data. Fig. 4.12e and 4.12f show that the resonance has serious effects on the synchronization. But if resonance can be avoided, the timestamping errors due to network conflicts are seriously reduced.

Summary

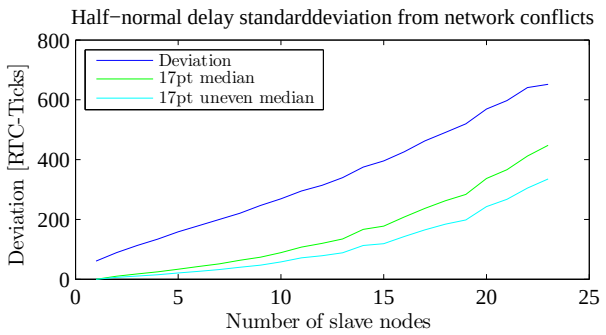
In summary, transmission delays which are not detected can be a source of big timestamping inaccuracies. Especially when network utilization rises the interference becomes common, which can be a problem for filters to deal with. Synchronizing data transmissions greatly reduces the



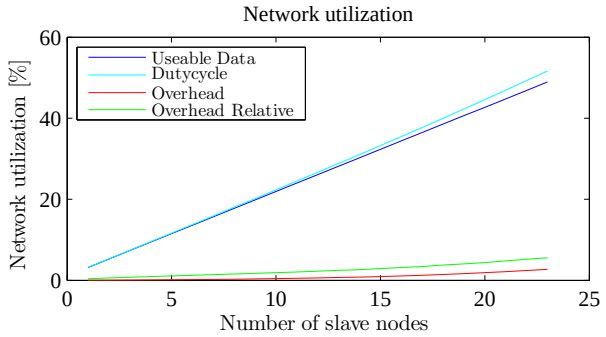
(a) Average delay of all timestamps



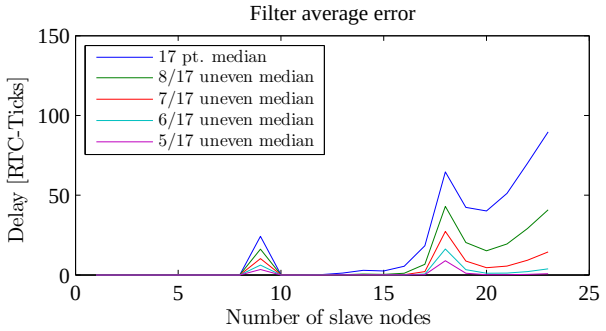
(b) Average delay, all timestamps, no random SYNC period



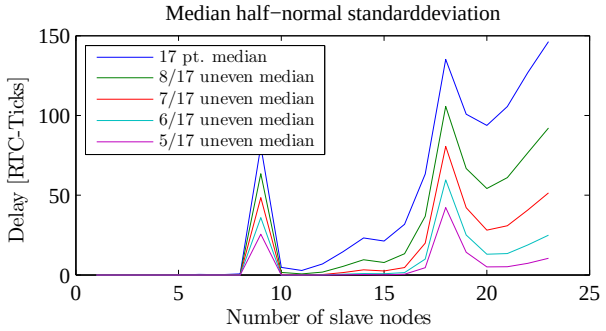
(c) Average deviation



(d) Duty Cycle



(e) Filtered SYNC delay



(f) Filtered SYNC delay deviation

Figure 4.12: Slotted mode delay, standard deviation and duty cycle

conflict potential, and therefore improves synchronization. However steps have to be taken that no sources can continuously interfere with each other. In chapter 5.3.6 filters will be analyzed to deal with this interference.

5 Improving Software based clock synchronization

This chapter presents improvements to software based clock synchronization. It gives an overview of the influence of the possible timestamping points, especially in relation to the challenges discussed in chapter 4 and explains briefly how the MULLE specific errors are compensated. The main part of this chapter concerns itself with the filtering of PTP data, resulting in accurate timestamps even if generated from erroneous data. These filters were both simulated in MATLAB and implemented on the MULLE platform. In this chapter filter simulations based on Raw PTP data measurements and network conflict simulations are conducted, while real world measurements of clock synchronization are presented in chapter 6. A section providing guidelines for filter selection rounds off this chapter.

5.1 System specific solutions

Peculiarities of the MULLE platform presented some unique challenges. This section presents a quick overview of the solutions, the challenges themselves are discussed in detail in chapter 4.

Avoiding MULLE RTC instabilities

As the MULLE RTC becomes unstable if triggered outside of a 512-tick grid unstable values were avoided. This is done by triggering on the closest valid clock value preceding the desired timestamp, and then delaying action until the required time has arrived. This does however result in increased interference by non-interruptable tasks (see section 4.3). Section 5.3 presents filter algorithms which can compensate the increased interference. The extra delay also severely limits the frequency where as tasks can be triggered. As in the current implementation only the burst voltage measurement had to be triggered once per second, this limitation could be accepted.

Compensating link asymmetry

As shown in chapter 4.2, with the timestamping point used limited link speed resulted in a static offset between the PTP Master and the PTP Slave. This offset has been calculated precisely and can be compensated by modifying the calculation of the offset

$$\Delta t_{MS,i} = (\Delta t_{SYNC,i} + \Delta t_{DELAY-REQ,i})/2 + \Delta t_{MESSAGE-LENGTH} \tag{5.1}$$

Due to this being a static offset error other calculation points are equally valid, as this compensation can be included at any point of the PTP offset calculation. Should the link speed not be known or other sources of asymmetry exist, [Exe14] gives an overview of methods that can be used to estimate this asymmetry factor. These methods do however require additional information not included in the basic synchronization messages, e.g. burst transmissions to estimate one-way link speed.

5.2 Compensation of Clock Frequency Offsets

A method to compensate this clock frequency offsets is proposed in IEEE 1588 Annex L[IS08] by calculating the frequency scale factor offset

$$F_n = \frac{(T_{1,n} - T_{1,n-1}) + (T_{1,n} + d_n - T_{2,n})}{T_{2,n} - T_{2,n-1}} \tag{5.2}$$

$$\delta_n = F_n - 1 \tag{5.3}$$

with

δ_n the frequency scale factor offset on receipt of the nth followup message

F_n the frequency scale factor on receipt of the nth followup message

$T_{1,n}$ the precise origin timestamp contained in the nth followup message

$T_{2,n}$ the reception timestamp of the nth sync message

d_n the sum of the current measured propagation time, the correction field of the nth sync message, and the correction field of the nth followup message

In a single-hop environment d_n can be reduced to the current measured propagation time. Unfortunately the jitter due to various reasons (see chapter 4) is far greater than the clock drift, therefore scaling the clock directly results in the local clock beginning to oscillate. Therefore this method is only applicable in implementations where a far better timestamping accuracy is available. Section 5.3.5 proposes a compensation scheme to overcome this limitation. While the basic compensation method used there is similar, it uses a packet based approach which can be optimized and integrated directly into the uneven median filter, reducing calculation times significantly.

5.3 Filtering PTP data

In this section several filter strategies to deal with these errors are discussed. While the filters presented make use of the unique properties of the errors models generated in chapter 4, they are also designed to smooth the normal jitter of the PTP messages. The ability to calculate how the errors occur does not allow the system to compensate automatically, as both the calculations and the synchronization would be too calculation intensive for the low-cost devices used. It does however greatly ease simulation to design and evaluate appropriate filters. As each filter builds on the knowledge generated by the filters before this section can also be seen as an evolution of filter techniques. The filters have been implemented on the MULLE devices, most evaluation has however been done in MATLAB. The data used for the simulation has been measured on the MULLE platform using the PTP raw measurement application.

As the filters use buffered previous timestamps to calculate their output, they require a certain amount of timestamp measurements to be available. To simplify implementation and improve the system reaction time, those buffers are filled with the first available measurement during filter initialization. This value represents the best estimation available at that time. However, should that measurement be erroneous that error is multiplied to the full buffer.

While it would be possible for some filtering schemes to dynamically scale both the buffer and the filters with the measurements available until the nominal buffer size has been reached, this would incur a big implementation overhead while only improving results during the startup phase. As the application only required measurements in an already synchronized network, this startup errors are accepted. At each filter it will be discussed how this startup error presents itself and after how many steps filter values should be considered valid.

General Equations

The following basic equations are used throughout the filters.

$$\Delta t_{SYNC,i} = (t_{SYNC,i} - t_{FOLLOWUP,i})/2 \tag{5.4}$$

$$\Delta t_{DELAY-REQ,i} = (t_{DELAY-RESP,i} - t_{DELAY-REQ,i})/2 \tag{5.5}$$

$$\Delta t_{MS,i} = (\Delta t_{SYNC,i} - \Delta t_{DELAY-REQ,i})/2 \tag{5.6}$$

with

$t_{SYNC,i}$	The local receiving timestamp of the ith SYNC packet
$t_{FOLLOWUP,i}$	The sending timestamp of the SYNC packet
$t_{DELAY-RESP,i}$	The receiving timestamp of the DELAY-REQUEST packet
$t_{DELAY-REQ,i}$	The local sending timestamp of the DELAY-REQUEST packet

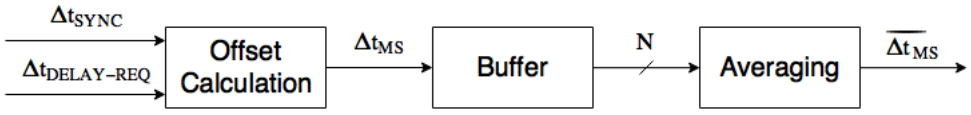


Figure 5.1: Averaging Filter

$\Delta t_{SYNC,i}$ The clock offset, calculated only from SYNC and FOLLOWUP i

$\Delta t_{DELAY-REQ,i}$ The clock offset, calculated only from DELAY-REQUEST and DELAY-RESPONSE i

$\Delta t_{MS,i}$ The offset between the local RTC and the PTP Master, compensated for transmission time

5.3.1 Averaging filter

The most basic filter, the averaging filter, filters data by collecting up to N datapoints and calculating the average delay from them. This can be done very efficiently by using $N_{=}2^n, n \in \mathbb{N}$, as this reduces the division to a bit-shift operation.

$$\overline{\Delta t}_{MS,i} = \frac{1}{N} * \sum_{k=0}^{N-1} \Delta t_{MS,i-k} \tag{5.7}$$

with

$\overline{\Delta t}_{MS,i}$ The filtered clock offset at SYNC-interval i

N The size of the averaging filter

Unfortunately, as fig. 5.2 shows, this approach does not work well with the error models that occur. While this filter does work well for a narrow-band gaussian error model, with the single large error spikes that occur each spike immediately results in a jump of the filtered value.

During initialization of the buffer the first value recorded is used to prefill the buffer, as this represents the best estimate available at the time. If this first measurement is erroneous the error incurred will slowly be reduced by $\frac{error}{N}$ by each step. Only after the whole buffer has been filled with different measurements should the filter output be considered valid.

5.3.2 Outlier rejection averaging filter

To combat these kinds of errors, outlier rejection filters can be added. Here the unique property that serious errors occur in spikes can be used. In a first step, outliers have to be identified. This is done by calculating the arithmetic mean and the standard deviation of the sample size. A similar algorithm is used as a popcorn spike suppressor in NTP [MMBK10].

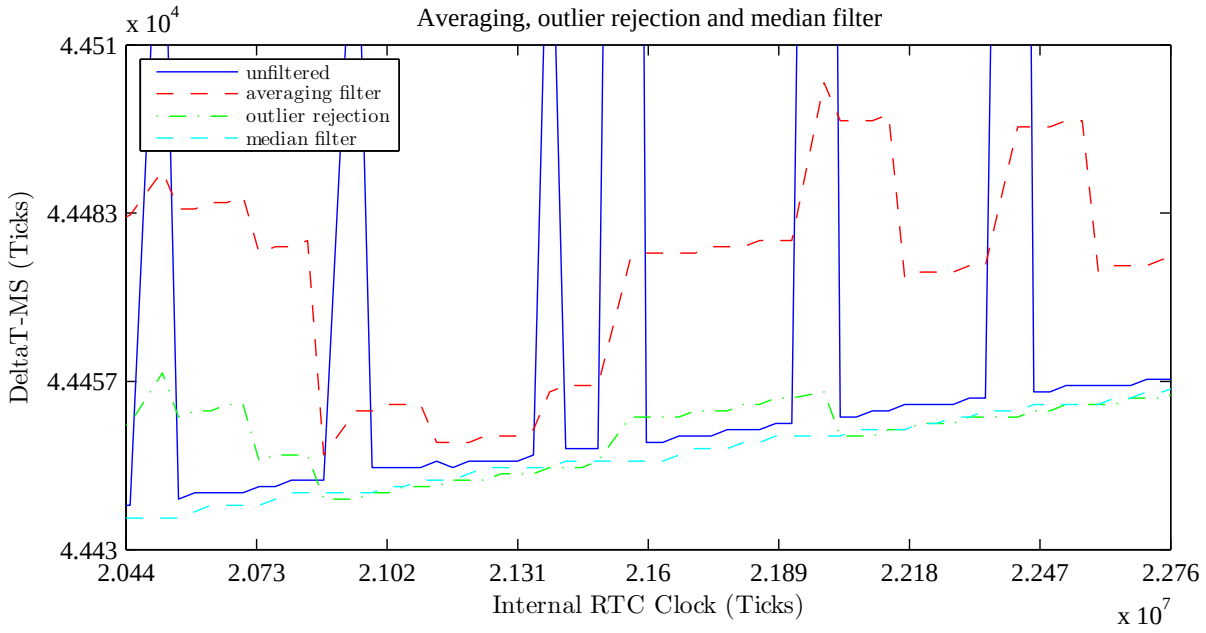


Figure 5.2: Comparison of averaging filter types

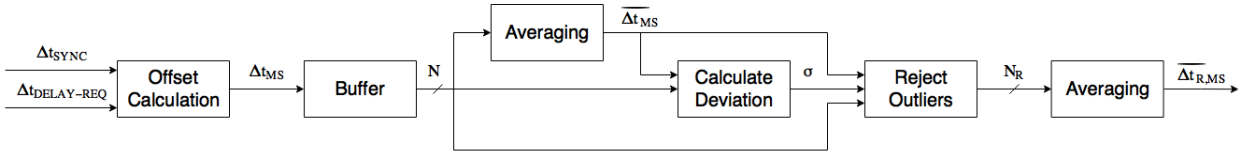


Figure 5.3: Outlier rejection averaging filter

$$\mu_i = \frac{1}{N} * \sum_{k=0}^{N-1} \Delta t_{MS,i-k} \tag{5.8}$$

$$\sigma_i = \sqrt{\frac{1}{N} * \sum_{k=0}^{N-1} (\Delta t_{MS,i-k} - \mu_i)^2} \tag{5.9}$$

As the range of regular jitter, which gets smoothed by averaging, may vary it is necessary to calculate both values. Afterwards an averaging filter, using only values which are within $l\sigma$ of the arithmetic mean, can be applied. This method allows the filter to dynamically adjust to varying conditions.

$$sum_i = \sum_{k=0}^{N-1} \begin{cases} \Delta t_{MS,i-k} & \text{if } \mid \Delta t_{MS,i-k} - \mu_i \mid < k * \sigma_i \\ 0 & \text{otherwise} \end{cases} \tag{5.10}$$

$$V_i = \sum_{k=0}^{N-1} \begin{cases} 1 & \text{if } \mid \Delta t_{MS,i-k} - \mu_i \mid < k * \sigma_i \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

$$\overline{\Delta tr}_{MS,i} = \frac{sum_i}{V_i} \tag{5.12}$$

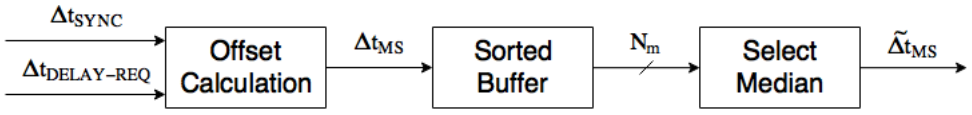


Figure 5.4: Median filter

with

μ_i	The arithmetic mean of the offset-measurements
σ_i	The standard deviation
l	The rejection filter limit. Should be set at least to 1, otherwise scenarios are possible where all measurements are rejected.
sum_i	The sum of the valid offset-measurements
V_i	The number of valid offset-measurements
$\overline{\Delta tr}_{MS,i}$	The rejection-filtered clock offset at SYNC-interval i
N	The size of the buffer

This solution offers significantly better results (fig. 5.2), however does not work well if multiple spikes happen within the measurement period. Especially a combination of large and small spikes allows the smaller spike to slip through, due to the changes in both the mean and the standard deviation caused by the larger spike. These patterns can be generated regularly by interfering non-interruptable tasks (4.3). Additionally the calculation time grew exponentially compared to the simple averaging filter. Due to the variable number of values which do not get rejected no optimization of the averaging division is possible, and the calculation of the square root is especially CPU intensive if no floating point unit (FPU) is available.

While further optimizations would be possible (e.g. using a variable rejection filter limit, which would keep the valid window as small as possible while still allowing multiple values for averaging) this path of filtering was discontinued in favor of median filters which reject outliers by default and are less CPU intensive.

The outlier rejection buffer is especially vulnerable to startup errors, as due to the erroneous result being replicated in the buffer, all other values get rejected as outliers. Therefore this filter will stay stuck at the erroneous value until, due to the new measurements, both mean and deviation have shifted to accept the new values. Normalization happens in 2 big steps, first when new values get accepted and second once the initial erroneous measurements get rejected, with a linear progression in-between due to the averaging nature.

5.3.3 Median filter

With the limited resources available, it is a far better solution to use a filter which rejects outliers by default. A running median smoother proves very effective in this regard [Arc05]. To filter the offset a sorted, monotonously increasing dataset

$$\Delta ts_{MS,i} = sort^{i}_{k=i-N_m+1}(\Delta t_{MS,k}) \tag{5.13}$$

is created with each new measurement. As spikes are either minima or maxima, the middle values of the dataset contain values unaffected by these kinds of errors. To calculate the median value of the dataset

$$\Delta\tilde{t}_{MS,i} = \begin{cases} \Delta ts_{MS,i}(\frac{N+1}{2}) & \text{if } N \text{ is odd} \\ \frac{1}{2} * (\Delta ts_i(\frac{N}{2}) + \Delta ts_i(\frac{N}{2}+1)) & \text{if } N \text{ is even} \end{cases} \tag{5.14}$$

with

$\Delta ts_{MS,i}$ The set of the last N offset-measurements, sorted monotonously by increasing value

$\Delta\tilde{t}_{MS,i}$ The filtered clock offset at SYNC-interval i

N The size of the median filter

has to be selected. This filter type deals far better with the occurring errors, as can be seen in fig. 5.2.

Calculation times can be reduced by using an uneven filter size, removing the need for averaging the two values around the median. Maintaining a sorted list of values of size N by removing the oldest element and inserting the new value at its appropriate place further optimizes the calculation by eliminating the need for sorting the whole list at each step.

Similar to the outlier rejection filter, during startup a median filter might get stuck on an erroneous measurement. Recovery however is usually rapid after $\frac{N}{2}$ steps, as all of the erroneous buffer values will get rejected at once.

5.3.4 Uneven median filter

The one-sidedness of the errors can further be used to improve accuracy by ignoring the u maximum values of the dataset, which most likely represent erroneous measurements. This can be implemented very simply by modifying the selection index of the median. If an additional factor $l = 2u$ is implemented this additionally allows the use of half-steps, allowing to switch between the even and odd calculation cases of eq. 5.16.

$$\Delta ts_{MS,i} = sort^{i}_{k=i-N_m+1}(\Delta t_{MS,k}) \tag{5.15}$$

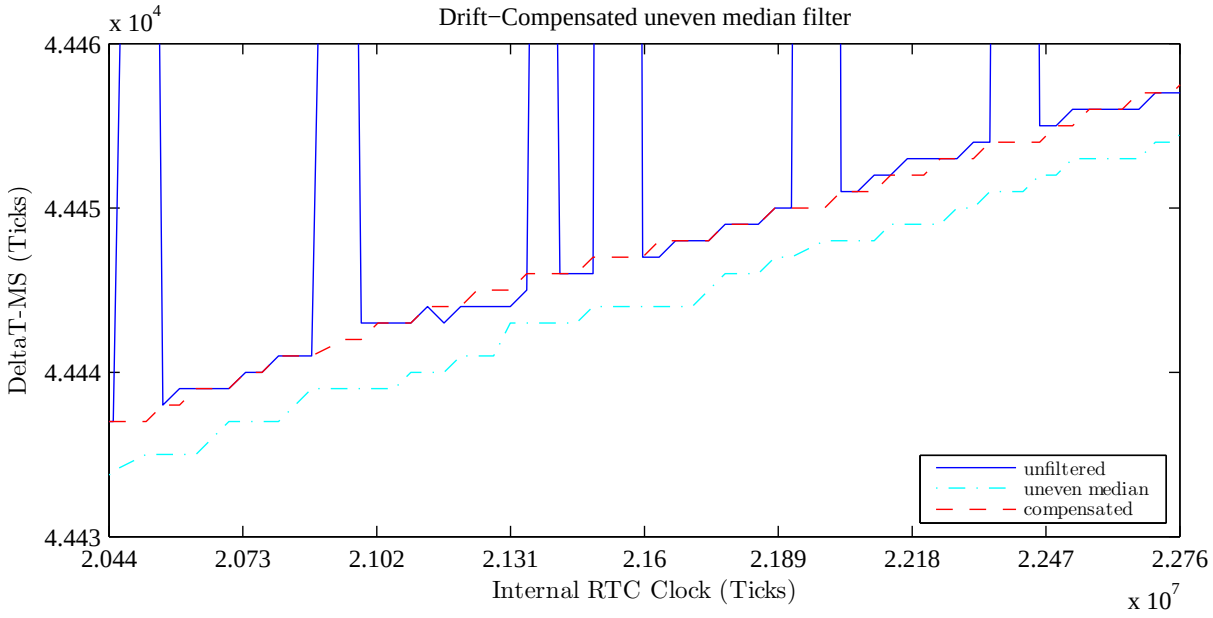


Figure 5.5: Comparison of median filter types

$$\Delta\tilde{t}_{MS,i} = \begin{cases} \Delta ts_{MS,i}(\frac{N_m+1+l}{2}) & \text{if } N_m + l \text{ is odd} \\ \frac{1}{2} * (\Delta ts_{MS,i}(\frac{N_m+l}{2}) + \Delta ts_{MS,i}(\frac{N_m+l}{2} + 1)) & \text{if } N_m + l \text{ is even} \end{cases} \tag{5.16}$$

This allows for tailoring the filter to the expected errors without increasing the calculation time. It is however important to note that moving the selection index too far greatly reduces the filtering capability for errors of the opposite polarity. Therefore it is crucial to test this filter with real data.

For the practical implementation an uneven median filter selecting the 7^{th} of $N_m = 17$ values has proven a good tradeoff between calculation time and filter performance, resulting in a 15 datapoint median after the rejection of $u = 2$ largest values.

Like the median filter this filter stays stuck at its initial value, and will recover in a single big jump from startup errors. Due to the unevenness it is however hard to predict the point when measurements become valid as this is dependent on the offset used and the polarity of the error in the initial measurement. Therefore measurements should only be considered valid after a full N_m steps have passed.

5.3.5 Drift compensated uneven median filter

If the linear clock drift between master and slave is greater than the jitter on the timestamps, the median filter can not work properly. To understand this limitation one has to analyze the uncompensated internal buffer of the filter in fig. 5.7. While outliers get rejected, the drift forces the median filter to select its results based on the slope and not based on the actual measurement jitter. This further results in an erroneous offset based on the drift and the filter size (fig. 5.5). By using a feedback loop to compensate the drift the measured offset values can be deskewed and the results greatly improved. To compensate this error first the average drift

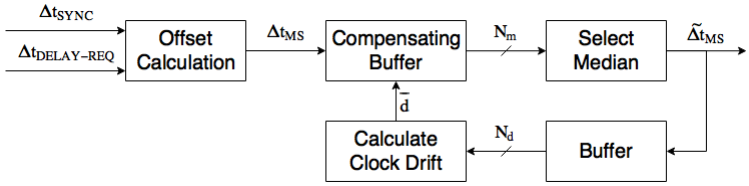


Figure 5.6: Compensated median filter

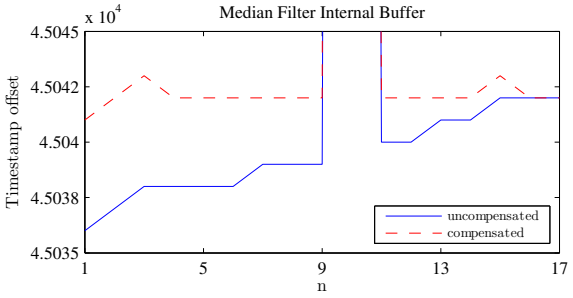


Figure 5.7: Drift compensation of median filter data

$$\overline{d}_i = \frac{\Delta\tilde{t}_{MS,i} - \Delta\tilde{t}_{MS,i-N_d+1}}{N_d} \tag{5.17}$$

has to be calculated by comparing the current and a previous filter value. This simple approach is possible because the quartz of the RTC clock can be assumed to be stable in the timescale of the drift filter history size N_d. This allows the calculation of a compensated clock offset

$$\Delta tc_{MS,k} = \Delta t_{MS,k} + \sum_{l=0}^{i-k-1} \overline{d}_{i-l} \tag{5.18}$$

which is represented in the compensated curve of fig. 5.7.

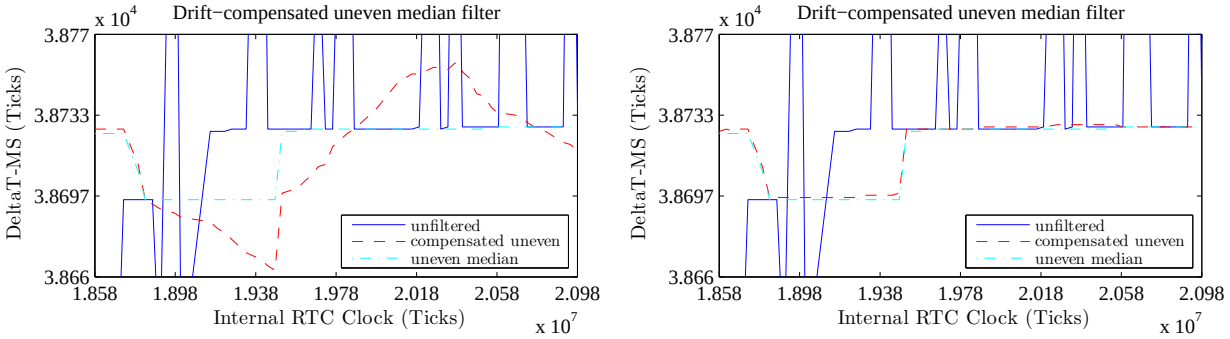
$$\Delta tcs_{MS,i} = sort^{i}_{k=i-N_m+1}(\Delta tc_{MS,k}) \tag{5.19}$$

$$\Delta\tilde{t}_{MS,i} = \begin{cases} \Delta tcs_{MS,i}(\frac{N_m+1+l}{2}) & \text{if } N_m + l \text{ is odd} \\ \frac{1}{2} * (\Delta tcs_{MS,i}(\frac{N_m+l}{2}) + \Delta tcs_{MS,i}(\frac{N_m+l}{2} + 1)) & \text{if } N_m + l \text{ is even} \end{cases} \tag{5.20}$$

By integrating the summation (5.18) with the buffer of the uneven median filter (5.19), calculations can be reduced to N_m additions at each step. Because the same compensating value is added to all elements in the buffer this operation does not break the sortedness of the buffer. Selecting $N_d = 2^{n_d}, n_d \in N$ further reduces calculation time, as divisions by powers of 2 can be done more efficiently by using bitshift operations.

In practice, compensating the clock offset (eq. 5.18) should only be enabled after at least $N_m + N_d$ calculation steps. Should the first measurement, used to prefill the buffer, be erroneous it takes up to N_m steps for the uneven median filter to provide accurate results. As the drift compensation should only be calculated from valid filter results, additional N_d steps are necessary for the compensation buffer to be filled with valid results. Otherwise, jumps in the frequency compensation

might lead to oscillating errors due to the feedback loop. While these errors are getting damped by the filter over time it is far better to avoid these oscillations in the first place.



(a) Unlimited drift rate

(b) Drift rate limited

Figure 5.8: Oscillation after uncompensated error in feedback loop

Situations in which the filtering process fails may also trigger oscillations. These are usually triggered by multiple consecutive erroneous measurements. Fig. 5.8a shows a situation in which multiple consecutive DELAY-REQUEST measurements have failed, with the error getting multiplied as 3 or 4 filter calculations take place between DELAY-REQUEST packets due to the more frequent SYNC packets. Because drift errors can be assumed to change slowly (e.g. due to temperature effects on quartz crystals), these errors can be prevented by limiting the change of the compensation factor to

$$\overline{d}_i \in [\,\overline{d}_{i-1} - \overline{d}_{max}, \overline{d}_{i-1} + \overline{d}_{max}] \tag{5.21}$$

As the average drift encountered in the experiments was around $10ppm$, a maximum change of $\overline{d}_{max} = 0.5ppm/step$ was used as a limit. This prevents the filter from oscillating (fig. 5.8b) while allowing it to compensate an arbitrary drift. This comes at the cost of longer initial drift compensation period, but this stabilizes before the feedback loop is activated. However, while this solution is simple and efficient, it requires an estimation of the magnitude of the drift.

5.3.6 Dual drift compensated uneven median filter

Conflicts on the shared medium may delay packet transmission, and therefore reception timestamps. This leads to an erroneous increase in the delay measurement (see 4.4). While median filters are very efficient at rejecting outliers, the fact that these errors are always positive can be used to further improve filter efficiency. However, this property gets lost once

$$\Delta t_{MS,i} = \frac{1}{2} * (\Delta t_{SYNC,i} - \Delta t_{DELAY-REQ,i}) \tag{5.22}$$

gets calculated, as errors in the $\Delta t_{DELAY-REQ,i}$ get counted negatively. Therefore the two delay measurements have to be filtered separately for optimal rejection. While clock drift has to be compensated the same way as in 5.3.5, it is only calculated from the more frequent SYNC packets, as a separate calculation for each filter would cause the internal filters to drift apart.

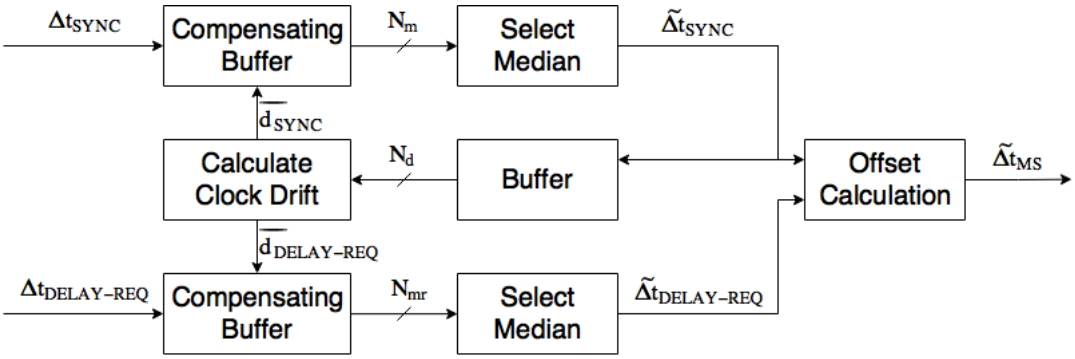


Figure 5.9: Dual drift compensated uneven median filter

$$\overline{d}_{SYNC,i} = \frac{\Delta\tilde{t}_{SYNC,i} - \Delta\tilde{t}_{SYNC,i-N_d+1}}{N_d} \tag{5.23}$$

Due to the DELAY-REQUEST packets arriving with a different frequency as the SYNC packets, this clock drift has to be scaled to conform with the average drift for DELAY-REQUEST packets.

$$\overline{d}_{DELAY-REQ,i} = \frac{T_{DELAY-REQ}}{T_{SYNC}} * \overline{d}_{SYNC,i} \tag{5.24}$$

This allows the compensation of the time offset values in the buffers.

$$\Delta tc_{SYNC,k} = \Delta t_{SYNC,k} + \sum_{l=0}^{i-k-1} \overline{d}_{SYNC,i-l} \tag{5.25}$$

$$\Delta tc_{DELAY-REQ,k} = \Delta t_{DELAY-REQ,k} + \sum_{l=0}^{i-k-1} \overline{d}_{DELAY-REQ,i-l} \tag{5.26}$$

As with the previous uneven median filter, values offset from the median by an offset factor l are selected. Due to the possibility that different filter sizes are used for SYNC and DELAY-REQUEST, a separate offset factor l_{mr} has to be defined for the second filter.

$$\Delta\tilde{t}_{SYNC,i} = \begin{cases} \Delta tc_{SYNC,i}(\frac{N_m+1+l}{2}) & \text{if } N_m + l \text{ is odd} \\ \frac{1}{2} * (\Delta tc_{SYNC,i}(\frac{N_m+l}{2}) + \Delta tc_{SYNC,i}(\frac{N_m+l}{2}+1)) & \text{if } N_m + l \text{ is even} \end{cases} \tag{5.27}$$

$$\Delta\tilde{t}_{DELAY-REQ,i} = \begin{cases} \Delta tc_{DELAY-REQ,i}(\frac{N_{mr}+1+l_{mr}}{2}) \\ \quad \text{if } N_{mr} + l_{mr} \text{ is odd} \\ \frac{1}{2} * (\Delta tc_{DELAY-REQ,i}(\frac{N_{mr}+l_{mr}}{2}) + \Delta tc_{DELAY-REQ,i}(\frac{N_{mr}+l_{mr}}{2}+1)) \\ \quad \text{if } N_{mr} + l_{mr} \text{ is even} \end{cases} \tag{5.28}$$

After filtering the current offset between PTP Master and Slave can finally be calculated.

$$\Delta\tilde{t}_{MS,i} = \frac{1}{2} * (\Delta\tilde{t}_{SYNC,i} - \Delta\tilde{t}_{DELAY_REQ,i}) \tag{5.29}$$

Due to the fact that SYNC and DELAY REQUEST have different periods two operating modes for this filter can be defined:

asynchronous mode In asynchronous mode, filter results are updated as soon as new data is available. This results in the best current offset estimation possible, but leads to filter values which may change in quick succession.

synchronous mode In synchronous mode the final calculation (eq. 5.29) acts as a latch. Filter result updates are tied to the more frequent SYNC packets, resulting in a constant period of filter value updates.

Selecting filter sizes

To avoid any synchronization errors between the filters both filters should cover the same time period.

$$T = (N_m - 1) * T_{SYNC} = (N_{mr} - 1) * T_{DELAY-REQ} \tag{5.30}$$

By selecting the closest multiple

$$N_{mr} = round((N_m - 1) * \frac{T_{SYNC}}{T_{DELAY-REQ}} + 1) \tag{5.31}$$

we get an appropriate filter size, with an error

$$\Delta T = \begin{cases} mod_{T_{DELAY-REQ}}((N_m - 1) * T_{SYNC}) & \text{if } <= \frac{T_{DELAY-REQ}}{2} \\ T_{DELAY-REQ} - mod_{T_{DELAY-REQ}}((N_m - 1) * T_{SYNC}) & \text{otherwise} \end{cases} \tag{5.32}$$

By adjusting the filter sizes to reflect the same period, equation 5.24 can also be simplified to correspond to filter sizes.

$$\overline{d}_{DELAY-REQ,i} = \frac{T_{DELAY-REQ}}{T_{SYNC}} * \overline{d}_{SYNC,i} \approx \frac{N_m}{N_{mr}} * \overline{d}_{SYNC,i} \tag{5.33}$$

With the values used in the implementation, $T_{DELAY-REQ} = 4s$ and $T_{SYNC} = 1.1s$, the following solutions are possible:

Table 5.1: Filter Sizes

SYNC Filter Size N_m	DELAY-REQ Filter Size N_{mr}	Period T	Period Difference ΔT
5	2	4.4	0.4
8	3	7.7	0.3
12	4	12.1	0.1
16	5	16.5	0.5
19	6	19.8	0.2
23	7	24.2	0.2

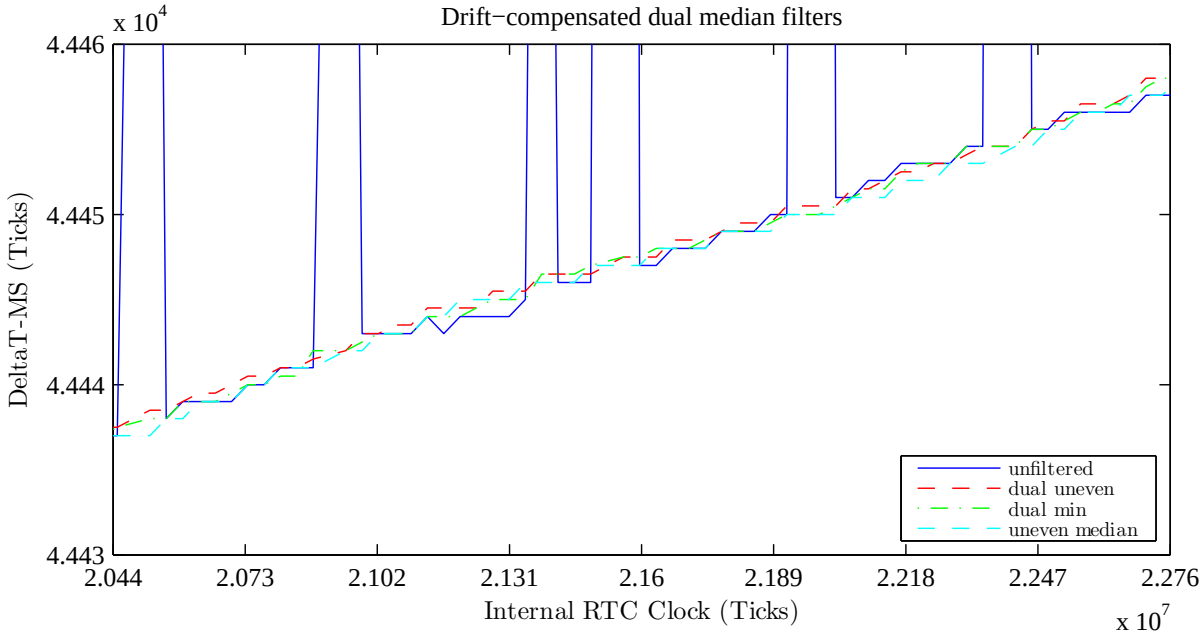


Figure 5.10: Comparison of dual median filter types

Simulation

For the simulation a synchronous mode filter, selecting the 7^{th} value of $N_m = 16$ (SYNC path) and the 2^{nd} of $N_{mr} = 5$ (DELAY-REQUEST path) was chosen. This filter size solution is closest to the filter parameters of the single-channel filters used previously. This minimizes differences from the filter window size and selection, highlighting the differences in filter structure. In figure 5.10 the performance of the dual filters can be seen in comparison to the single-channel drift-compensated uneven median filter. Additionally the dual minima selection filter (see section 5.3.6) is shown. In this application scenario the difference to the single-channel drift compensated uneven median filter (section 5.3.5) remains minimal. However, when confronted with network traffic the advantages of the dual filter become apparent (fig. 5.11). The ability to filter both channels separately allows it to be far more resilient to errors.

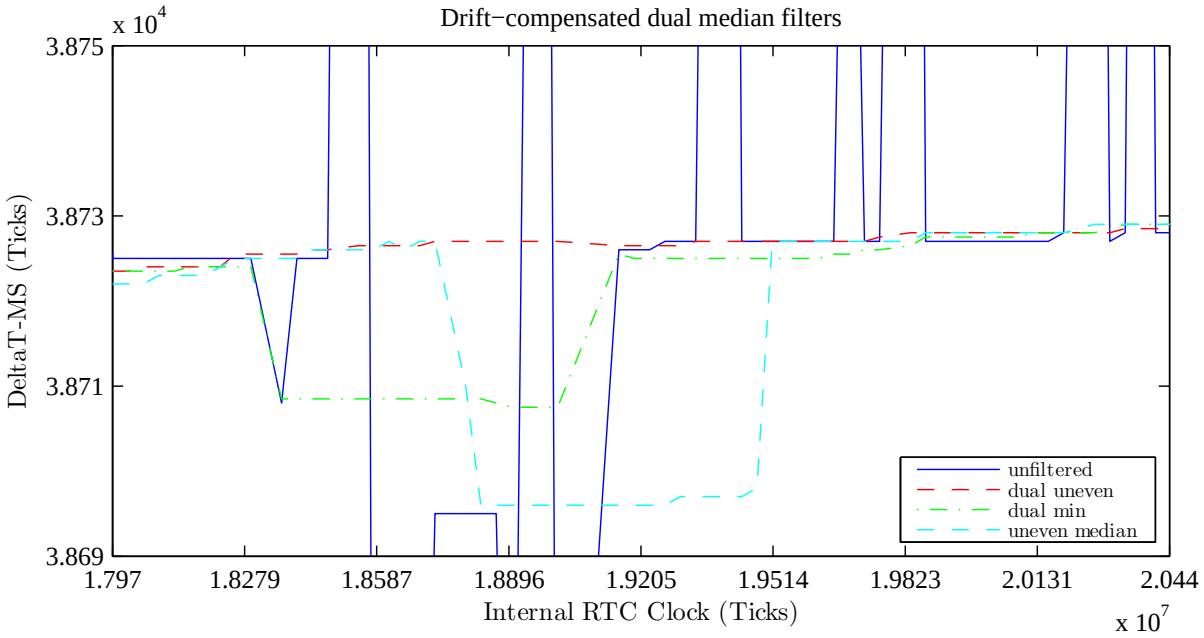


Figure 5.11: Comparison of dual median filter types with network conflicts

Dual minima selection filter

A special case of the dual channel filter is the dual minima selection filter. This filter is based on the assumption that, due to causality, all factors influencing delay can only increase the total delay of a transmission. As non-deterministic effects get compounded on the transmission delay, selecting the minimum delay measurement should remove all non-deterministic influences. This is similar to the filtering algorithm used in NTP [MMBK10]. For simulation a dual filter selecting the 1^{st} value of $N_m = 16$ (SYNC path) and of $N_{mr} = 5$ (DELAY REQUEST path) was used, shown in figures 5.10 and 5.11.

In regular cases the filter performs similar to both the regular dual drift-compensated uneven median filter, and due to the minima selection can outperform the median filtering when the main source of timestamping errors is heavy traffic on the shared medium. However, while all delays are positive, compared to the average delay of a non-deterministic source negative spikes can occur. Figure 5.11 begins its error pattern with a negative spike on the SYNC delay measurement of 17 ticks. The minima selection filter immediately responds and stays at an erroneous value until the spike leaves its buffer of $N_m = 16$ values. Using only these spikes as a synchronization source by increasing buffer length is not an option as they occur very rarely and randomly. The exact source of this $500\mu s$ delay difference could not be determined, but filter types which do select an (uneven) median are effective at compensating negative erroneous delay measurements.

5.4 Selecting the right filter

While it has been shown that both accuracy and precision can be greatly improved by using appropriate filters, there is always a tradeoff between resource usage (processing time and memory usage) and accuracy. Therefore to select the right filter the intended system has to be evaluated. As the median filter has been implemented with minimal extra overhead over the averaging filter while providing greatly improved results only median filters are recommended. Because filter selection is a multidimensional problem (required accuracy and precision, available resources, expected interfering processes and network conflicts, etc.) only general guidelines can be presented here. For accurate predictions filter simulations have to be conducted.

One important factor to decide is whether the expected duty cycle on the network will lead to significant conflicts, or if internal error sources are more likely. If network conflicts are a main source of errors, then a dual-filter approach can improve results due to its splitting of the problem. This allows for uneven filters to effectively deal with the errors. However due to the need to maintain two distinct filters this almost doubles resource usage. As DELAY-REQUEST errors are highly unlikely to result from internal errors, if a low duty cycle is selected a single-channel filter will provide good results.

Filter sizes should be chosen depending on the amount of erroneous timestamping measurements that should be rejected. Here the longer period of DELAY-REQUEST measurements becomes dominant, as less data is available. In a dual-filter, with the $4s$ DELAY-REQUEST period this results in a minimum filter size of $8s$, if only the lower value is considered. In a single-channel filter it should be assumed that at least the lowest 3.64 measurements are due to an erroneous DELAY-REQUEST measurement, therefore the minimum filter size is 7 for a median filter. This minimum size however precludes the use of an uneven median filter. As unevenness improves filter performance larger filter sizes are recommended.

Drift compensation becomes necessary if the accuracy error incurred due to filter sizes becomes problematic. This problem is compounded by increasing filter size, as older values may be selected. The maximum error due to drift can be estimated by $\Delta t_{Drift,max} = 2 * \delta_{Clock} * T_{SYNC} * N_{Filter}$, as it is possible that 2 clocks with the accuracy δ_{Clock} may drift into opposite directions.

6 Evaluation

While simulations are extremely useful during development, a system should also perform in the real world. Therefore the developed compensation strategies have been implemented, and in this chapter measurements of the synchronization accuracy are presented. Additionally a short section evaluating the ARROWHEAD framework from a developers point of view is included.

6.1 Synchronization Accuracy

Synchronization accuracy was measured in different ways: An external synchronized toggling pin allowed for external measurements with an oscilloscope between two nodes. Also, by calculating the voltage zero crossing point of the distributed power line measurement multiple nodes could be evaluated, provided they were measuring the same power line. This led to three scenarios:

- Accuracy between the PTP Master and the PTP Slave, measured by oscilloscope
- Accuracy between two PTP Slaves, measured by oscilloscope
- Accuracy over a distributed measurement network consisting of PTP Slaves, measured by data

To describe synchronization accuracy, two factors have to be measured: The average offset between the nodes, and the standard deviation.

$$\overline{\Delta t} = \frac{1}{n}\sum_{i=1}^{n} t_{2,i} - t_{1,i} \tag{6.1}$$

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\Delta t_i - \overline{\Delta t})^2} \tag{6.2}$$

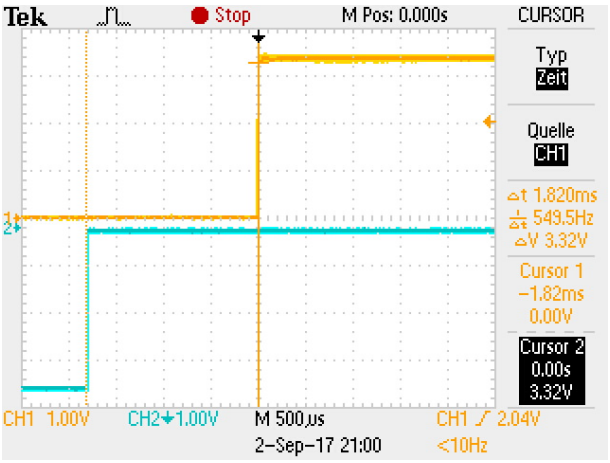


Figure 6.1: Offset due to asymmetric packet length

6.1.1 External measurement between PTP Master and PTP Slave

These measurements were done with only the basic PTP software running, avoiding interference from conflicting tasks. During initial measurements a static offset of $1.82ms$ was measured, but could be compensated in software by an addition of 59 RTC ticks. Analysis of this offset error led to the discovery of the inherent asymmetry described in section 4.2. A filtered version of the uncompensated measurement can be seen in figure 6.1, all other measurements were taken with the offset compensated.

The pictures were created by using the infinite afterglow display setting on the Tektronix TDS 2022B oscilloscope, starting the measurement after $60s$ to allow for initial filter convergence and measuring for an additional $180s$.

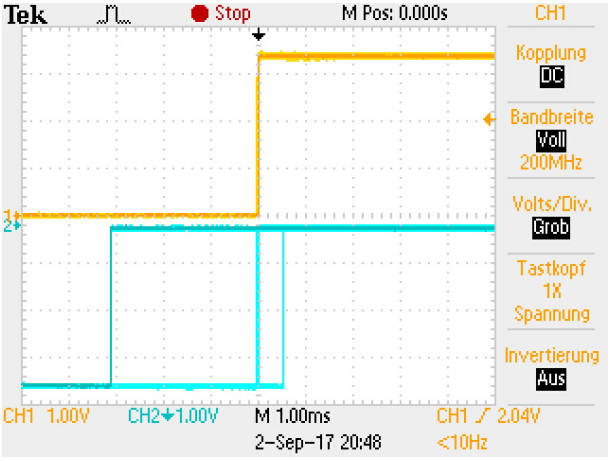
Table 6.1: Master Slave Accuracy

	Min $min(\Delta t)$	Max $max(\Delta t)$	Offset $\overline{\Delta t}$	Deviation σ	Comment
Unfiltered	$-3160\mu s$	$520\mu s$	$16.8\mu s$	$511.33\mu s$	No filter
Averaged	$140\mu s$	$310\mu s$	$218.6\mu s$	$52.72\mu s$	16 point averaging filter
Median	$-20\mu s$	$26.4\mu s$	$2.37\mu s$	$11.57\mu s$	17 point drift-compensated uneven median filter

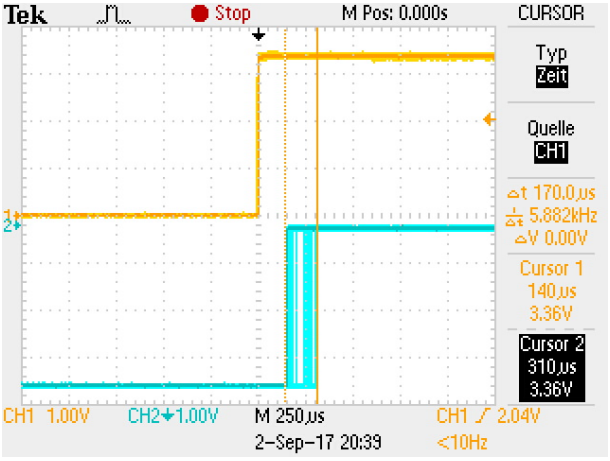
The unfiltered PTP data has serious outliers, with errors in the millisecond range (fig. 6.2a). The averaging filter fares a lot better, however it introduces a big offset. This is due to the uncompensated RTC-clock drift of the averaging filter (fig. 6.2b). If a continuous clock drift of d is assumed the averaging over multiple values results in an offset error of

$$\Delta\overline{t_{MS,drift}} = d * \frac{N-1}{2} \tag{6.3}$$

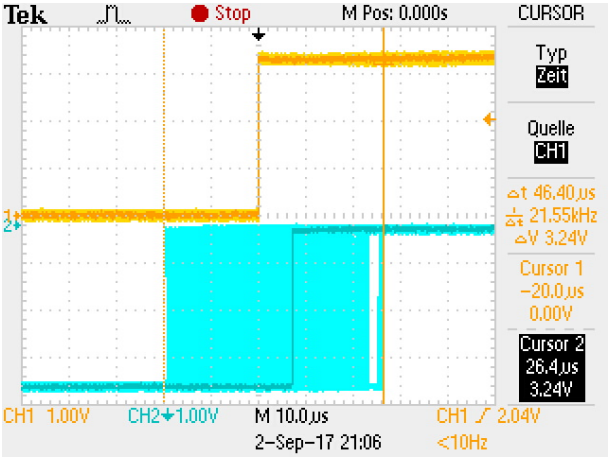
Due to the drift compensation integrated in the uneven median filter this error can be avoided, resulting in a synchronization accuracy within 1 RTC tick of the master clock (fig. 6.2c).



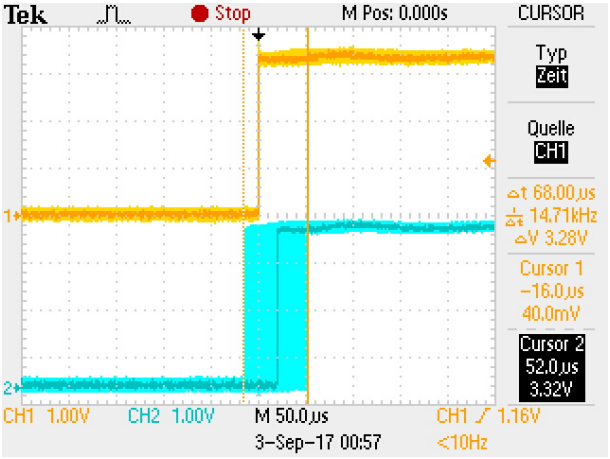
(a) Master-Slave unfiltered



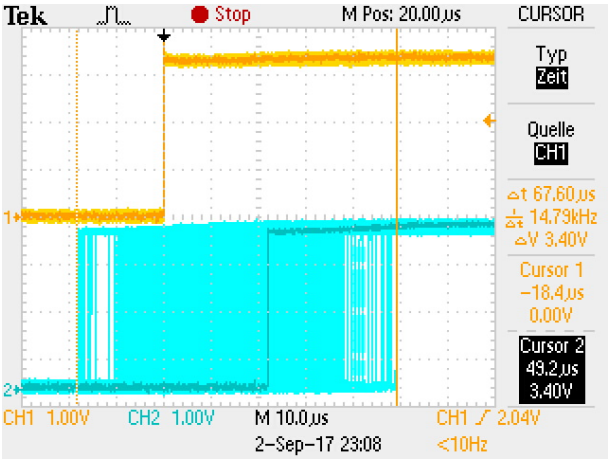
(b) Master-Slave averaged



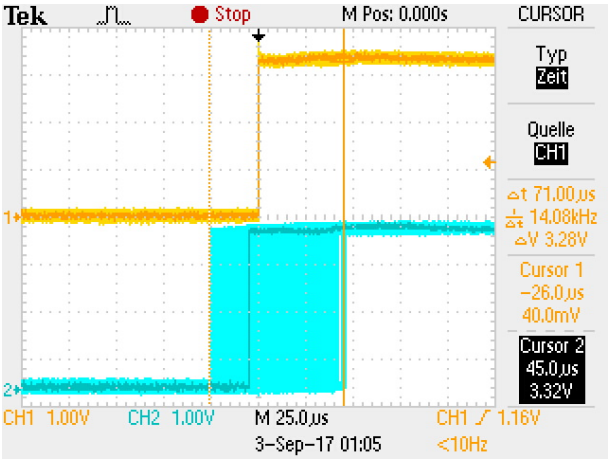
(c) Master-Slave median filter



(d) Slave-Slave averaged



(e) Slave-Slave median filter



(f) Slave-Slave median filter without asymmetry

Figure 6.2: Oscilloscope filter measurements

6.1.2 External measurement between two PTP Slaves

The simple expectation would be that due to multiple synchronization error sources the synchronization would be worse. This assumption however does discount that some error sources affect both slaves, therefore these errors are not uncorrelated. If both nodes are affected the same way this error is compensated if only comparing the clocks of the slave nodes. Therefore slave-slave synchronization in the unfiltered and the averaged measurements are better than the master-slave accuracies with the same filters. In this configuration the averaging filter provided quite accurate results fig. (6.2d). However, simulations and PTP raw measurements show that this holds only true as long as the errors are correlated. Once the extra errors of the non-interruptable measurement task (chapter 4.3) are included performance drops rapidly. Using the median filter (fig. 6.2e), which effectively removes these errors present does the assumption of both nodes as uncorrelated error sources hold true. As expected, turning off the asymmetry compensation (fig. 6.2f) does not change the results, as the same error occurs on both slave nodes.

Table 6.2: Slave Slave Accuracy

	Min $min(\Delta t)$	Max $max(\Delta t)$	Offset $\overline{\Delta t}$	Deviation σ	Comment
Unfiltered	$-982\mu s$	$640\mu s$	$-10.09\mu s$	$306.54\mu s$	No filter
Averaged	$-16\mu s$	$52\mu s$	$13\mu s$	$19.87\mu s$	16 point averaging filter
Median	$-18.4\mu s$	$49.2\mu s$	$13.59\mu s$	$16.02\mu s$	8/17 drift-compensated uneven median filter

6.1.3 Analysis of voltage measurement data

To simulate a realistic measurement environment the ADC of multiple MULLE nodes was used to measure a rectified power line voltage $V_{in} = |230V * \sqrt{2} * sin(50Hz * 2\pi * t)|$. All nodes are measuring the same voltage source which allows to calculate the clock synchronization accuracy from the measured voltage data itself. Figure 6.3 shows the measurement setup, each MULLE node uses the the extension board (see 3.2.2) to rectify the input and adjust the voltage level. To provide realistic background traffic and collect the measurement data, each MULLE registered itself at the ARROWHEAD service discovery, and is sending their data to the Leshan server in the background where data is analyzed stored in the MySQL database. The measurement data consists of a set of measured voltage values V_i and the measurement timestamps t_i. The following algorithm is used to calculate the zero crossing timestamp from the voltage data:

First a local minima has to be found

$$\exists i | (V_{i-1} > V_i) \wedge (V_{i+1} > V_i) \tag{6.4}$$

Near the zero crossing of the measured voltage sine signal, the following simplification can be assumed

$$\lim_{t->0} \frac{sin(t)}{t} = 1 \tag{6.5}$$

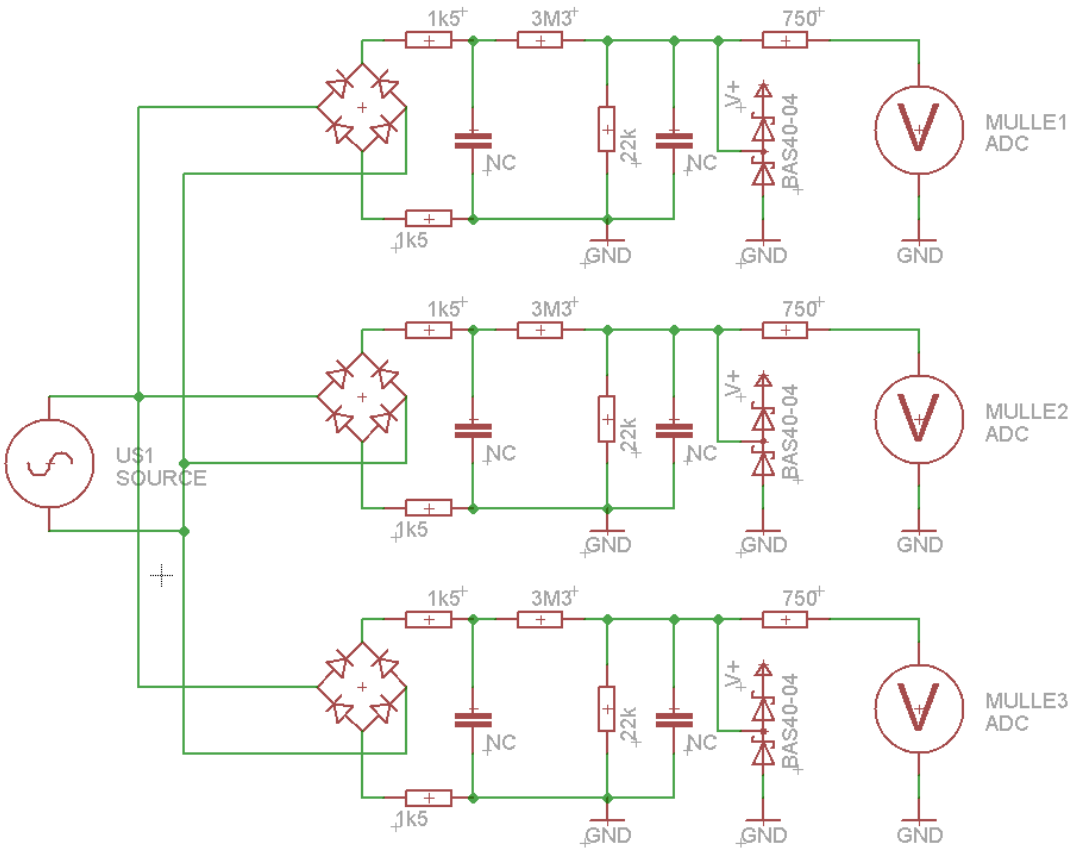


Figure 6.3: Distributed voltage measurement

Therefore linear interpolation is applicable to calculate the zero crossing time

$$t_{V=0} = t_i + \frac{v_i}{k} \tag{6.6}$$

with the voltage differential

$$k = \lim_{t->t_{V=0}} \frac{dV}{dt} = \begin{cases} \frac{V_{i+1}+V_i}{t_{i+1}-t_i} & \text{if } V_{i-1} \geq V_{i+1} \\ \frac{V_1+V_{i-1}}{t_{i-1}-t_i} & \text{if } V_{i-1} < V_{i+1} \end{cases} \tag{6.7}$$

With this algorithm the point of the zero crossing can be calculated for each measured dataset, and synchronziation between the devices evaluated.

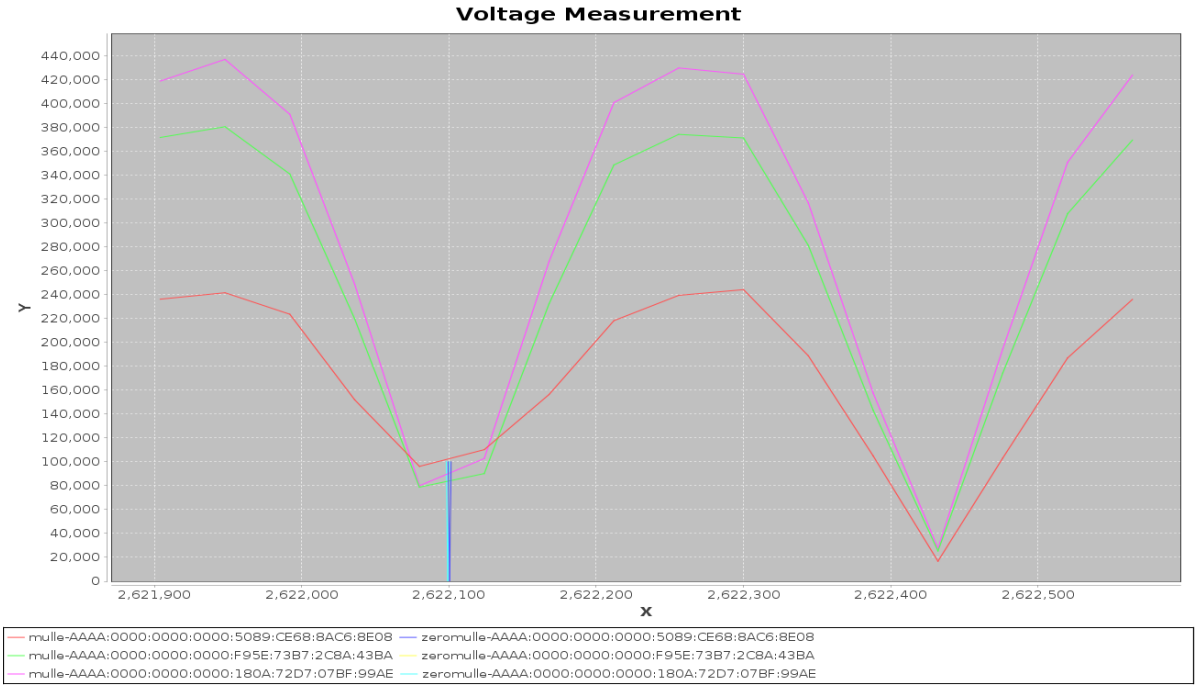


Figure 6.4: Zero crossing calculation at LESHAN server

Without a fixed reference clock all values are now calculated against the mean of each measurement zero crossing set. Therefore an average offset statistic becomes meaningless. The other values are calculated by:

$$\overline{t_{ZERO,i}} = \frac{1}{N}\sum_{k=1}^{N} t_{ZERO,i,k} \tag{6.8}$$

$$\Delta t_{ZERO,i} = t_{ZERO,i,k} - \overline{t_{ZERO,i}}; \tag{6.9}$$

$$\sigma_i = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(t_{ZERO,i,k} - \overline{t_{ZERO,i}})^2} \tag{6.10}$$

Table 6.3 shows the calculated synchronization accuracies. It has to be noted that due to the calculation method any measurement errors of the analog measurement additionally influence the calculation. The calculation method assumes perfect analog to digital conversion, however real ADCs have non-linear errors, especially when approaching the GND or V_{cc} voltages, which may skew results. Fig. 6.4 shows one such measurement, and while all 3 nodes are in sync it can be seen that the ADCs are not evenly calibrated. One error that generated several outlier measurements was due to the wrong selection of the zero crossing. As the measurement covered a full sine period multiple zero crossings exist in the data and slight differences in the data may result in a different selection of the first minimum. This resulted in the large maximum error calculation of $6551\mu s$ where a detection error of a half-period resulted in one zero calculation being $10ms$ apart from the other 2 results. As this does not constitute a clock synchronization error, these measurement errors were compensated by moving the erroneous value by $10ms$ in post-processing, resulting in the filtered values.

Table 6.3: Zero crossing measurement accuracy

	Min $min(\|t_{ZERO,i}\|)$	Max $max(\|t_{ZERO,i}\|)$	Deviation $\overline{\sigma}$	Comment
3 slaves	$0\mu s$	$6551\mu s$	$220\mu s$	8/17 point uneven median
3 slaves	$0\mu s$	$468\mu s$	$98.9\mu s$	corrected zero crossing selection

6.2 Synchronization Overhead

Table 6.4: Synchronization overhead

	Length *Byte*	Frequency *s*	Bits per Second *bit/s*	Percent %
SYNC multicast	76	1.1107	547.4	0.547
FOLLOWUP multicast	76	1.1107	547.4	0.547
DELAY-REQUEST unicast	120	4.0107	239.4	0.239
DELAY-RESPONSE unicast	120	4.0107	239.4	0.239
STATIC data			1094.8	1.0948
DYNAMIC per slave			478.7	0.4787

Table 6.4 shows the data used for synchronization. The percentage calculations are based on the $100kbit/s$ 6LoWPAN settings used in the current network. While the static data, resulting from the multicast packets from the master, does not overly tax the network, the dynamic traffic per PTP Slave can be troublesome. In chapter 4.4 the influence of network conflicts is discussed in detail, and due to the impossibility to detect network conflicts with the available timestamping point this can dramatically decrease synchronization precision. Therefore this limits both the maximum number of slaves and the available bandwidth for application data.

6.3 ARROWHEAD

This section gives a short overview of the state of the ARROWHEAD framework. Many of the problems stem from the developing nature of the framework at the 1.9.2016 cutoff date. This led to the implementation of the system to be challenging, resulting in multiple workarounds. However, once the whole system was up and running, even radical changes to the configuration could be implemented relatively easily. Therefore the proof of concept is a success, but at the time of implementation the framework was not yet at a mature development stage.

Challenges encountered include, but are not limited to:

- Because every team had their own focus, a somewhat insular design approach could be discerned. For example the ARROWHEAD CENTOS server officially should run on IPv4 only, whereas the MULLE platform for measurements was using 6LoWPAN and therefore using IPv6 only.

- The Service Discovery Proxy officially supported both COAP and HTTP. However, the COAP implementation was unstable and crashed often, therefore a COAP-HTTP bridge (CROSSCOAP) was used to keep the system stable.
- Due developing nature, even ARROWHEAD support on the MULLE platform was limited. While an OMA-LwM2M client was implemented, at that development stage communication with the Service Discovery had to be implemented by manually manipulating COAP/XML packets.

Therefore, from a developer point of view the author agrees with the official statement: *The Arrowhead Framework is designed as a SOA Reference Architecture (RA). The current solution of this RA should be seen as a proof of concept for the Arrowhead Framework (and its partner applications)* [VBF+17].

7 Conclusion and Outlook

Summary

A distributed measurement system according to the ARROWHEAD framework has been implemented and evaluated. The hardware platform for this development was the MULLE platform, which offers support for many ARROWHEAD functions for orchestration. This measurement system consists of several measurement nodes, a router which connects them to an external network where multiple servers are located, which handle service discovery and measurement orchestration. Due to the service oriented architecture employed, the nodes can automatically configure themselves to work as a distributed measurement network.

The main challenge of the distributed measurement system was to accurately synchronize the clocks within the network. The protocol used for synchronization was PTP, where the router additionally acted as a PTP Master. Due to a lack of hardware support, software timestamping had to be used. This led to multiple unique challenges, which are the focus of this work. These challenges have been analyzed, simulated, and appropriate filters were designed to overcome these limitations.

One limitation of software only timestamping is that sending and receiving packets cannot be timestamped at exactly the same point. Due to the fact that they are buffered at the transceiver, packets could only be timestamped before they were sent and after they were fully received. Therefore sent packets are timestamped at the beginning of the frame, while received packets are timestamped at the end. If all synchronization packets would have the same length that would not constitute a problem as PTP would automatically compensate for any symmetric delays. However due to the header compression used in 6LoWPAN this does not hold true, as the multicast SYNC packets are compressed better than the DELAY-REQUEST unicast packets sent by other nodes. It has been shown that this results in a static clock synchronization error, which can be compensated.

Software solutions are also limited by the fact that the processor cannot be interrupted at all times. Some high priority tasks (e.g. measurements) do not allow interrupts while they are performing their work. This prevents software timestamping until the blocking high priority task

is completed, leading to an erroneous timestamp. If these tasks occur regularly this creates an interference pattern which can hamper synchronization accuracy. This error was analyzed and a mathematical model for simulations was created, leading to the development of specialized filters based on median smoothing. The results have already been published at the ETFA 2017 conference.

Finally, due to the fact that a packet can only be timestamped before it is passed to the transceiver, transmission delays due to busy networks cannot be detected instantly. This results in erroneous transmission timestamps. These problems increase with network utilization as packet transmission gets delayed more frequently and for longer periods of time. Simulations of the impact of network utilization on clock synchronization accuracy have been conducted.

It has been discussed how these challenges would interact with different timestamping implementations and solutions for the challenges discovered have been presented. Multiple filter models have been analyzed to find optimal solutions which can account for regular synchronization challenges like noise and clock drift as well as compensate these unique problems. These filters have been optimized to reduce their calculation time. This led to the development of the dual uneven drift-compensated median filter. Based on these improvements guidelines for filter selection, and their limits, have been created.

With these solutions in place it has been shown that accurate distributed measurements can be gathered without using specialized hardware. This has been done by conducting synchronized measurements of a single voltage signal, with the data automatically gathered by a LESHAN server to be analyzed. Due to calculation of the zero crossing of the voltage, synchronization accuracy can be inferred. This clock synchronization accuracy has been independently verified by using external oscilloscope measurements.

Conclusion and outlook

Synchronizing a network based solely on software timestamping includes several major unique challenges. This work does not claim that hardware timestamping does not provide big advantages. However, even without specialized hardware, high accuracy synchronized measurements are possible. The unique properties of these timestamping errors can then be used for appropriate filter design, allowing for accurate clock synchronization based on imperfect data. This allows for software only solutions in many distributed applications. This reduces both cost and development time, as software solutions can be ported far easier to different systems than hardware solution. By using the models generated in this work it can also be estimated prior to implementation whether a software solution will be sufficient or if a hardware solution is necessary. Therefore this work can be used to improve future synchronized systems, playing its small part in the development of a networked world. As a first step towards this goal, parts of this work have been published at the 22^{nd} IEEE conference on Emerging Technologies And Factory Automation [MSPS17].

The precision time protocol (PTP) was chosen as it was a generic synchronization protocol, whose basic principle, the round trip delay measurement, is used in a multitude of clock synchroniza-

tion schemes. However, it is not optimized for multi-hop wireless environments. It is therefore recommended to combine the methods generated with available synchronization protocols, allowing them to deal with imperfect conditions and removing the need for hardware timestamping. TSPN would be a simple fit due to its similarities with the PTP protocol while its generated synchronization tree is optimized for multi-hop networks. As this would additionally reduce the synchronization packet length, overhead and network conflicts would also improve.

Alternatively, combining this work with a receiver-receiver synchronization protocol like RBS would remove the most challenging influence of software timestamping errors: network access conflicts. This would therefore be best suited to higher level timestamping. The line fitting algorithm in RBS could be replaced by the filter techniques developed in this work, which have both good performance and are optimized for resource constrained systems. This would allow a real time implementation of RBS in IoT devices. The requirement however is that RBS only works in broadcast domains with at least 3 nodes, therefore it would need to be complemented by another synchronization protocol. Here PTP would be the best candidate for point-to-point links.

For use in general networks the PTP implementation would have to be extended to include the full best clock selection algorithm. While the PTP implementation is written to be as generic as possible, a few parts are rooted in the CONTIKI operating system. These would have to be replaced by wrappers for true portability.

While the filters presented deliver very good performance they are still susceptible to packet loss because they are designed as a per-step approach. An outlier detection algorithm on the reception timestamps might detect these errors and allow the system to compensate for them by interpolating. In addition, filter performance when dropping some conditions (e.g. periodicity for the non-interruptable task) has to be evaluated.

Further work is necessary to study these possibilities.

8 Appendix

List of Figures

List of Tables

Literature

[14905] IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 15.1a: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Wireless Personal Area Networks (WPAN). In: *IEEE Std 802.15.1-2005 (Revision of IEEE Std 802.15.1-2002)* (2005), June, S. 1–700

[45708] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. In: *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)* (2008), July, S. 1–269

[74616] IEEE Standard for Low-Rate Wireless Networks. In: *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)* (2016), April, S. 1–709

[77816] IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. In: *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)* (2016), Dec, S. 1–3534

[79217] IEEE Standard for Information technology–Telecommunications and information exchange between systems - Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Sub 1 GHz License Exempt Operation. In: *IEEE Std 802.11ah-2016 (Amendment to IEEE Std 802.11-2016, as amended by IEEE Std 802.11ai-2016)* (2017), April, S. 1–594

[ABB+14] Adame, Toni ; Bel, Albert ; Bellalta, Boris ; Barcelo, Jaume ; Oliver, Miquel: IEEE 802.11 ah: the WiFi approach for M2M communications. In: *IEEE Wireless Communications* 21 (2014), Nr. 6, S. 144–152

[Arc05] Arce, Gonzalo R.: *Nonlinear signal processing: a statistical approach*. John Wiley & Sons, 2005

[AS05] Akyildiz, IW S. ; Su, IW: Time-diffusion synchronization protocols for sensor networks. In: *IEEE/ACM Transactions on Networking* (2005), S. 1626–1645

[BFK+14] Blomstedt, Fredrik ; Ferreira, Luis L. ; Klisics, Markus ; Chrysoulas, Christos ; de Soria, Iker M. ; Morin, Brice ; Zabasta, Anatolijs ; Eliasson, Jens ; Johansson, Mats ; Varga, Pal: The arrowhead approach for soa application development and documentation. In: *Industrial Electronics Society, IECON 2014-40th Annual Conference of the IEEE* IEEE, 2014, S. 2631–2637

[BHG+13] Baccelli, Emmanuel ; Hahm, Oliver ; Gunes, Mesut ; Wahlisch, Matthias ; Schmidt, Thomas C.: RIOT OS: Towards an OS for the Internet of Things. In:

Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on IEEE, 2013, S. 79–80

[BM02] Bush, Randy ; Meyer, David: Some Internet architectural guidelines and philosophy. (2002)

[Bra89] Braden, Robert: Requirements for Internet hosts-communication layers. (1989)

[BV09] Buratti, Chiara ; Verdone, Roberto: Performance analysis of IEEE 802.15. 4 non beacon-enabled mode. In: *IEEE Transactions on Vehicular Technology* 58 (2009), Nr. 7, S. 3480–3493

[Com16] Combs. *Wireshark network protocol analyzer*. `https://www.wireshark.org/`. 2016

[CON12] CONTIKI. *timesynch.h*. `http://contiki.sourceforge.net/docs/2.6/a01741.html`. 2012

[CON16] CONTIKI. *soc-rtc.c*. `https://github.com/contiki-os/contiki/blob/master/cpu/cc26xx-cc13xx/dev/soc-rtc.c`. 2016

[DGV04] Dunkels, Adam ; Gronvall, Bjorn ; Voigt, Thiemo: Contiki-a lightweight and flexible operating system for tiny networked sensors. In: *Local Computer Networks, 2004. 29th Annual IEEE International Conference on* IEEE, 2004, S. 455–462

[DÖH07] Dunkels, Adam ; Österlind, Fredrik ; He, Zhitao: An adaptive communication architecture for wireless sensor networks. In: *Proceedings of the 5th international conference on Embedded networked sensor systems* ACM, 2007, S. 335–349

[Ecl16] Eclipse. *Leshan*. `http://www.eclipse.org/leshan/`. 2016

[EGE02] Elson, Jeremy ; Girod, Lewis ; Estrin, Deborah: Fine-grained network time synchronization using reference broadcasts. In: *ACM SIGOPS Operating Systems Review* 36 (2002), Nr. SI, S. 147–163

[Erl08] Erl, Thomas: *SOA design patterns*. Pearson Education, 2008

[Exe14] Exel, Reinhard: Mitigation of asymmetric link delays in IEEE 1588 clock synchronization systems. In: *IEEE Communications Letters* 18 (2014), Nr. 3, S. 507–510

[FFR+11] Ferrari, P. ; Flammini, A. ; Rinaldi, S. ; Bondavalli, A. ; Brancati, F.: Evaluation of timestamping uncertainty in a software-based IEEE1588 implementation. In: *2011 IEEE International Instrumentation and Measurement Technology Conference*, 2011. – ISSN 1091–5281, S. 1–6

[FGK11] Freris, Nikolaos M. ; Graham, Scott R. ; Kumar, PR: Fundamental limits on synchronizing clocks over networks. In: *IEEE Transactions on Automatic Control* 56 (2011), Nr. 6, S. 1352–1364

[GKS03] Ganeriwal, Saurabh ; Kumar, Ram ; Srivastava, Mani B.: Timing-sync protocol for sensor networks. In: *Proceedings of the 1st international conference on Embedded networked sensor systems* ACM, 2003, S. 138–149

[GOP12] Gomez, Carles ; Oller, Joaquim ; Paradells, Josep: Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. In: *Sensors* 12 (2012), Nr. 9, S. 11734–11753

[Har15] Hartke, Klaus: Observing resources in the constrained application protocol (CoAP). (2015)

[IBM16] IBM. *Crosscoap*. `https://github.com/ibm-security-innovation/crosscoap`. 2016

[IS02] Instrumentation, IEEE ; Society, Measurement: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. In: *IEEE Std 1588 TM -2002* (2002)

[IS08] Instrumentation, IEEE ; Society, Measurement: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems.

In: *IEEE Std 1588 ™ -2008* (2008)

[JVE+04] Johansson, Jonny ; Völker, Matthias ; Eliasson, Jens ; Östmark, Åke ; Lindgren, Per ; Delsing, Jerker: MULLE: a minimal sensor networking device: implementation and manufacturing challenges. In: *IMAPS Nordic Annual Conference: 26/09/2004-28/09/2004* International Microelectronics and Packaging Society, Nordic chapter, 2004, S. 265–271

[METS17] Mahmood, Aneeq ; Exel, Reinhard ; Trsek, Henning ; Sauter, Thilo: Clock Synchronization Over IEEE 802.11—A Survey of Methodologies and Protocols. In: *IEEE Transactions on Industrial Informatics* 13 (2017), Nr. 2, S. 907–922

[MKHC07] Montenegro, Gabriel ; Kushalnagar, Nandakishore ; Hui, Jonathan ; Culler, David: Transmission of IPv6 packets over IEEE 802.15. 4 networks. 2007

[MLJ+15] Moreira, Naiara ; Lázaro, Jesús ; Jimenez, Jaime ; Idirin, Mikel ; Astarloa, Armando: Security mechanisms to protect IEEE 1588 synchronization: State of the art and trends. In: *Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS), 2015 IEEE International Symposium on* IEEE, 2015, S. 115–120

[MMBK10] Mills, D ; Martin, J ; Burbank, J ; Kasch, W: RFC 5905: Network Time Protocol Version 4: Protocol and Algorithms Specification. Internet Engineering Task Force (IETF), 2010. ht tp. In: *tools. ietf. org/html/rfc5905* (2010)

[MO83] Marzullo, Keith ; Owicki, Susan: Maintaining the time in a distributed system. In: *Proceedings of the second annual ACM symposium on Principles of distributed computing* ACM, 1983, S. 295–305

[Mon16] Montori, Federico. *Simple Service Discovery.* `https://forge.soa4d.org/svn/arrowhead/Common%20Design%20Repository/01.%20WORKING_PROPOSAL/05.%20Prototypes/SimpleServiceDiscovery%5bproxy%5d/`. 2016

[MSPS17] Mitaroff-Szécsényi, Jeronimo ; Priller, Peter ; Sauter, Thilo: Compensating Software Timestamping Interference from Periodic Non-Interruptable Tasks. In: *Proc. of the ETFA 2017* IEEE, 2017

[Pos80] Postel, Jon: User datagram protocol. 1980

[Pos81a] Postel, Jon: Darpa Internet Protocol Specification / STD 5, RFC 791, September. 1981

[Pos81b] Postel, Jon: Transmission control protocol. (1981)

[RAK10] Rahamatkar, Surendra ; Agarwal, Ajay ; Kumar, Narendra: Analysis and comparative study of clock synchronization schemes in wireless sensor networks. In: *Analysys* 2 (2010), Nr. 3, S. 536–541

[Rom88] Romkey, JL: RFC 1055: Nonstandard for transmission of IP datagrams over serial lines: SLIP. In: *Chapter 11 Networking* (1988)

[SHB14] Shelby, Zach ; Hartke, Klaus ; Bormann, Carsten: The constrained application protocol (CoAP). (2014)

[She12] Shelby, Zach: Constrained RESTful environments (CoRE) link format. (2012)

[Sta96] Standardization, I: ISO/IEC 7498-1: 1994 information technology–open systems interconnection–basic reference model: The basic model. In: *International Standard ISOIEC* 74981 (1996), S. 59

[SV03] Sichitiu, Mihail L. ; Veerarittiphan, Chanchai: Simple, accurate time synchronization for wireless sensor networks. In: *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE* Bd. 2 IEEE, 2003, S. 1266–1273

[TH09] Treytl, Albert ; Hirschler, Bernd: Security flaws and workarounds for IEEE 1588 (transparent) clocks. In: *Precision Clock Synchronization for Measurement, Control*

and Communication, 2009. ISPCS 2009. International Symposium on IEEE, 2009, S. 1–6

[TH11] Thubert, Pascal ; Hui, Jonathan W.: Compression format for IPv6 datagrams over IEEE 802.15. 4-based networks. (2011)

[Tia12] Tian, Linyi: Lightweight m2m (oma lwm2m). In: *OMA device management working group (OMA DM WG), Open Mobile Alliance (OMA)* (2012)

[VBF+17] Varga, Pal ; Blomstedt, Fredrik ; Ferreira, Luis L. ; Eliasson, Jens ; Johansson, Mats ; Delsing, Jerker ; de Soria, Iker M.: Making system of systems interoperable–The core components of the arrowhead framework. In: *Journal of Network and Computer Applications* 81 (2017), S. 85–95

[WM10] Wobschall, Darold ; Ma, Yuan: Synchronization of wireless sensor networks using a modified IEEE 1588 protocol. In: *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2010 International IEEE Symposium on* IEEE, 2010, S. 67–70

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, am November 4, 2017 _____

Jeronimo Govinda Mitaroff-Szécsényi