



TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology



Clock Synchronization in a Distributed Hardware-in-the-Loop Testbed

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Embedded Systems

by

Thomas Reisinger, BSc

Registration Number 01125966

to the Faculty of Electrical Engineering and Information Technology

at the TU Vienna

Advisor: Privatdoz. Dipl.-Ing. Dr.techn. Wilfried Steiner

Assistance: Dr. Michael Steurer

Dr. Mark Stanovich

Dr. Karl Schoder

Vienna, 1st June, 2022

Thomas Reisinger

Wilfried Steiner



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I would like to express my sincere gratitude to Vienna University of Technology and Florida State University (FSU) Center for Advanced Power Systems (CAPS) for this cooperative master's thesis. In the course of my studies and many working hours in the study rooms of the Central Information Team (CID), a very familiar study group has established itself over the years. The thus self-proclaimed "CID-Room-Gang" has meanwhile become not only my closest circle of friends but also very valued work colleagues who were always up for a scientific discussion round. I would also like to thank my father, Michael Reisinger, through whose support over the years, I was able to focus on my studies. No matter how tiring and exhausting the studies were at times, there was one person in particular who always stood behind me and whose emotional support I could always rely on, thank you Marlies Metzich. A big thank you also goes to my advisors Wilfried Steiner, Mark Stanovich, and Michael (Mischa) Steurer who supervised me during my thesis. Last but not least, I would like to thank all my friends and family who are not mentioned by name and who always stood behind me and supported me during my studies.

This work was sponsored in part by the US Office of Naval Research (ONR) under grant number N00014-16-1-2956.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Power Hardware-in-the-Loop (PHIL) und Controller Hardware-in-the-Loop (CHIL) Experimente sind wertvoll für die Forschung und Entwicklung neuartiger Energiesystemtechnologien. In Kombination mit Netzwerkemulatoren wie CORE (Common Open Research Emulator) können komplexe Simulationen mit minimalen Hardwareanforderungen durchgeführt werden. Solche Simulationen sind jedoch recht komplex, da die Anzahl der gleichzeitigen Ereignisse und der erzeugten Daten steigt. Daher ist es wichtig, eine gemeinsame Uhr zur Verfügung zu stellen, mit der die Daten mit einem Zeitstempel versehen werden können. Zu diesem Zweck können genaue Zeitverteilungsmechanismen wie das Präzisionszeitprotokoll (PTP) IEEE-1588 genutzt werden. Das Center for Advanced Power Systems (CAPS) an der Florida State University (FSU) ist führend auf dem Gebiet der HIL-Simulationen. Derzeit umfasst die Infrastruktur des CAPS und der Partneruniversitäten für die Co-Simulation keine GPS-synchronisierten Hochpräzisions-Taktsynchronisationsgeräte, die in einem HIL-Experiment verwendet werden. Daher ist es das Ziel dieser Masterarbeit, ein GPS-Taktsynchronisationskonzept für das HIL-Testbed zu entwickeln. Das entwickelte Konzept wird ermöglichen, dieses PTP-Backbone mit verschiedenen Komponenten eines HIL-Testbeds bei CAPS zu verwenden und die Vorteile einer gemeinsamen Zeitbasis zwischen unabhängigen Hardwarekomponenten zu verifizieren. Dies ermöglicht die Messung der Einweg-Verzögerung zwischen zwei Geräten mit einer Auflösung im Nanosekundenbereich. Diese Methode kann insbesondere dazu verwendet werden, das Verhalten von emulierten Netzwerken in CORE im Vergleich zu realen Netzwerken zu überprüfen. Darüber hinaus kann die Auflösung von zeitgestempelten Daten aus Echtzeitsimulatoren (RTS) für eine genauere Ereignisbestimmung erhöht werden.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Power Hardware-in-the-Loop (PHIL) and Controller Hardware-in-the-Loop (CHIL) experiments are valuable for the research and development of novel power-system technologies. In combination with network emulators like CORE (Common Open Research Emulator) complex simulations can be realized with minimal hardware requirements. However, such simulations are quite complex as the number of simultaneous events and generated data grows. Therefore, it is important to provide a common clock by which to timestamp data. For this, accurate timing distribution mechanisms, like the precision time protocol (PTP) IEEE-1588 can be leveraged. The Center for Advanced Power Systems (CAPS) at Florida State University (FSU) is a leader in the field of HIL simulations. Today, the infrastructure at CAPS and partner universities for co-simulation does not include GPS synchronized high precision clock synchronization cross devices used in a HIL experiment. It is, thus, the objective of this master's thesis to develop a GPS clock synchronization concept for the HIL Testbed. The developed concept will demonstrate the possibility to use that PTP Backbone with different components of a HIL Testbed at CAPS and verify the benefits of a common time base between independent hardware components. This enables the opportunity to measure one-way path delay between two devices in nanosecond resolution. In particular, this method can be used to validate the behavior of emulated networks in CORE compared to real networks. In addition, the resolution of time stamped data from real-time simulators (RTS) can be increased for more precise event determination.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung	v
Abstract	vii
Contents	ix
List of Figures	xi
List of Tables	xiii
Acronyms	xv
1 Introduction	1
1.1 Thesis Contributions	2
1.2 Description of the research content	2
1.3 Expected Outcome	3
2 State of the Art	5
2.1 Precision Time Protocol IEEE 1588	5
2.1.1 Clock Types	5
2.1.2 Clock Synchronization	6
2.1.3 Precision Time Protocol (PTP) Profile IEC 61850-9-3	7
2.2 Hardware in the Loop Simulations	9
2.3 Case Studies	10
2.3.1 GPS Clock Synchronization Accuracy	10
2.3.2 Clock Synchronization in HIL Systems	12
2.3.3 Common Open Research Emulator (CORE)	14
2.4 Current Hardware in the Loop Testbed	16
2.4.1 Center for Advanced Power Systems	16
2.4.2 Power Hardware in the Loop Testbed	16
2.4.3 Controller Hardware in the Loop Testbed	18
3 System Architecture / Implementation	21
3.1 PTP Compatibility of HIL Interfaces	21

3.1.1	Real Time Digital Simulator	21
3.1.2	Linux Operating System	23
3.1.3	Non PTP Compatible Devices	24
3.2	PTP Backbone	26
3.2.1	Grandmaster Clock	26
3.2.2	PTP Switch	28
3.2.3	PTP Hardware Configuration	28
3.2.4	Final Hardware Setup	35
4	System Performance Validation	39
4.1	User Datagram Protocol (UDP) measurement script	40
4.1.1	UDP Socket Configuration	42
4.1.2	Sending and receiving UDP packets	42
4.1.3	Accessing timestamps via a control messages	44
4.1.4	Linuxptp data	44
4.2	CORE Validation Process	45
4.2.1	CORE Measurement Setup	45
4.2.2	CORE Validation Results	46
4.3	RTDS Validation Process	54
4.3.1	RTDS Measurement Setup	54
4.3.2	RTDS Validation Results	56
4.4	SyncBox validation Process	59
4.4.1	SyncBox Measurement Setup	59
4.4.2	SyncBox Validation Results	60
5	Outlook	65
5.1	Conclusion	65
5.2	Future Work	66
	Bibliography	67

List of Figures

2.1	Example peer delay mechanism between master and slave clock [1].	6
2.2	Peer-to-Peer mechanism with a transparent clock [2].	8
2.3	The Basic structure of PHIL/CHIL-based experimental test [3].	10
2.4	OSI layer topology and an illustrated Controller Hardware-in-the-Loop (CHIL) setup [4].	10
2.5	A concept overview to integrate IEEE 1588 power profile into a Power Hardware-in-the-Loop (PHIL) testbed [5].	13
2.6	The proposed software-based time synchronization mechanism [5].	13
2.7	The block diagram of the experimental setup [5].	14
2.8	Overview of a representative CHIL setup to evaluate distributed energy management control system performance [6].	15
2.9	Response time between a) a real hardware switch and b) an emulated switch within Common Open Research Emulator (CORE) [6]	16
2.10	Current Power Hardware in the Loop Testbed setup overview.	17
2.11	Florida State University (FSU)-Center for Advanced Power Systems (CAPS) Power Hardware in the loop Amplifiers setup.	18
2.12	FSU-CAPS Controller Hardware in the loop setup.	19
3.1	Extended CHIL testbed with a PTP backbone.	22
3.2	Real Time Digital Simulator (RTDS) periphery including different interface cards [7].	23
3.3	Core components of the PTP Backbone.	36
4.1	Measurement setup to validate system performance.	40
4.2	Flowchart of the UDP measurement script	41
4.3	Measurement setup during the validation process.	45
4.4	Emulated switch within the CORE simulation.	46
4.5	One-way latency of direct Ethernet connection, influenced of uncalibrated state using hardware timestamps.	47
4.6	One-way latency of direct Ethernet connection, influenced of uncalibrated state using software timestamps.	48
4.7	Examples of possible clock drift, or path delay measurement accuracy without PTP synchronization.	49
4.8	Comparison of one-way path delay over three different hardware switches.	51
		xi

4.9	Deviation of the propagation delay of an emulated switch according to different network parameters.	53
4.10	Schematic of the RSCAD UDP communication using the GTSYNC card as a time reference.	54
4.11	Visualization of an example GTNET UDP measurement including all monitoring signals.	56
4.12	Example latency measurement setup between GTNET-SKT and Linux machine.	56
4.13	Deviation of path delay between Optiplex 9010 and RTDS.	58
4.14	Influence of the timestep on the measured path delay.	59
4.15	Measurement setup for SyncBox clock stability.	60
4.16	Example of the rising edge detection algorithm.	61
4.17	Min/Avg/Max clock drift compared to reference Channel 1.	61
4.18	Minimal, average, and maximal drift compared to Channel 1 during 40 min period.	62
4.19	Average drift compared to Channel 1 during the total measurement period.	63
4.20	Drift between 10 MHz output ports within the SyncBox.	63

List of Tables

2.1	72-hour H-Maser linear fit residuals.	12
3.1	Overview of PTP features of a GTSYNC card.	22
3.2	Overview of PTP features of Linuxptp.	24
3.3	Hardware requirements for a Network Interface Card (NIC) to communicate with IEEE 1588 PTP [8]	24
3.4	Oscillators integrated in Meinberg Devices [9]	25
3.5	Mandatory (M) and optional (O) requirements for the PTP Backbone	26
3.6	Protocols and profiles of the microSync RX102 Grandmaster (GM) [10]	27
3.7	Different antenna types which are compatible to the Meinberg receiver [10]	27
3.8	MAC and IP addresses of all components within the PTP network	29
3.9	List of available Global Positioning System (GPS)/Galileo satellites.	30
3.10	Meinberg SyncBox PTP configuration.	30
3.11	Flag description of ptp4l and phy2sys command.	32
3.12	Example output from ptp4l and phc2sys command.	33
3.13	GM microSync RX102 description of status LEDs [10].	34
3.14	Description of status LEDs regarding Meinberg SyncBox.	35
3.15	All components used or available.	37
4.1	Comparison of path delay results depending on measurement method.	50
4.2	Comparison of one-way path delay over three different hardware switches.	51
4.3	Average propagation delay of an emulated switch according to different network parameters.	52
4.4	Difference between the real and emulated switch in CORE.	52
4.5	Path-delay between GTNET and Optiplex machine, depending on RTDS timestep.	57
4.6	Summary of internal and external drifts of two SyncBoxes.	64



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acronyms

- 1PPS** one Pulse per Second. 10–12, 14, 21, 22, 25, 65, 66
- ADC** Analog Digital Converter. 12
- BC** Boundary Clock. 5–7
- BMCA** Best Master Clock Algorithm. 5, 7, 26
- CAN** Controller Area Network. 19, 35, 66
- CAPS** Center for Advanced Power Systems. xi, 2, 16, 18, 19, 21, 24–26
- CHIL** Controller Hardware-in-the-Loop. xi, 9, 10, 14–16, 18, 22, 35
- CORE** Common Open Research Emulator. xi, xiii, 1, 3, 14–16, 19, 39, 40, 45, 46, 50, 52, 65
- CSU** COMPISO System Unit. 18
- DAC** Digital Analog Converter. 12
- DUT** Device Under Test. 1, 9, 12, 14, 17–19, 24, 26, 35, 59, 66
- E2E** End-to-End. 5, 25, 26
- FPGA** Field Programmable Gate Array. 12, 22
- FSU** Florida State University. xi, 2, 16, 18, 19, 21, 26
- GM** Grandmaster. xiii, 7, 21, 26, 27, 33, 34
- GMC** Grandmaster Clock. 2, 5, 7, 9, 12, 14, 21, 24, 26, 28, 31, 32, 34, 40, 44, 45, 59, 62
- GNSS** Global Navigation Satellite Systems. 5
- GPS** Global Positioning System. xiii, 5, 10–12, 21, 25–27, 30, 31, 33, 34, 65, 66

GUI Graphical User Interface. 22, 28–30, 34

HIL Hardware-in-the-Loop. 1–3, 9, 16, 18, 19, 21, 23, 25, 26, 39, 59, 65, 66

HMI Human-Machine Interface. 10, 16, 17

IEC International Electrotechnical Commission. 8

IF Intermediate Frequency. 11, 27

IP Internet Protocol. 10, 19, 28, 42, 43, 55

IRGBH Inter Rack Global Bus Hub. 23

IRIG Inter-Range Instrumentation Group. 25, 65

MAC Media Access Control. 28

MMC Multilevel Converters. 17, 18

NIC Network Interface Card. xiii, 14, 24, 31–34, 39, 40, 42, 45, 48

NTP Network Time Protocol. 5, 12, 17, 25, 29, 43, 59

OS Operating System. 23, 32, 65

P2P Peer-to-Peer. 5–7, 9, 22, 25, 26, 29, 30

PHIL Power Hardware-in-the-Loop. xi, 2, 9, 12, 13, 16–18, 35, 66

PoE Power over Ethernet. 25, 28

PPB Part per Billion. 32

PPO Programmable Pulse Output. 25, 26

PTP Precision Time Protocol. ix, xi, xiii, 1–3, 5, 7–9, 12, 14, 21–26, 28–36, 39, 40, 44, 45, 49, 50, 52, 55–57, 59, 60, 62, 65, 66

PTPd PTP daemon. 23

PUP Power Utility Profile. 8

RTDS Real Time Digital Simulator. xi–xiii, 15–18, 21–23, 39–41, 55–58, 65

RTS Real Time Simulator. 1, 9, 12, 15, 16, 21, 65

RTT Round Trip Time. 3, 45, 48, 49, 52

- SFP** Small Form-factor Pluggable. 26, 28
- SNTP** Simple Network Protocol. 31
- SPAN** Switched Port Analyzer. 30
- TAI** International Atomic Time. 29, 31–33, 54
- TC** Transparent Clock. 5–7
- TCP** Transmission Control Protocol. 10, 19
- TTL** Transistor-Transistor Logic. 25
- UDP** User Datagram Protocol. x–xii, 15, 22, 39–46, 48, 49, 54–57
- UTC** Coordinated Universal Time. 5, 12, 29, 31, 32, 55
- VVS** Variable Voltage Sources. 17, 18



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Introduction

Every device during the development process requires a variety of simulation and testing procedures before integration into the field. Since software-based simulation includes assumptions and simplifications, Hardware-in-the-Loop (HIL) simulations offer a more realistic approach. HIL simulations can be used for developing and testing real hardware under real circumstances. This technique reduces costs and over hardware testing, and includes testing with real-world issues like hardware weakness, noise, or randomly triggered events[5].

In combination with such simulation, two major components are used, a Real Time Simulator (RTS) and the Device Under Test (DUT). In addition, virtual network emulators can be used to further compute large-scale network analysis[11]. A RTS is responsible for the major computation and simulation process. Events of a simulation are executed sequentially during discrete time periods given by the RTS processor. Therefore, a discrete event counter starts with every simulation and can be used as a time reference for the simulation. Depending on the specific use case a virtual network emulator CORE can be utilized to create a network topology between RTS and DUT [6]. This reduces the amount of necessary peripheral hardware like switches, routers, or computers. Thus, emulated networks improve the controllability of network components and reduce costs by replacing real hardware.

Instead of discrete timing events from RTS, a higher resolution of time source can be achieved with a high precision GPS synchronized reference. Therefore a clock distribution system is required to synchronize all components of the simulation with the same time reference. IEC 61850 is a widely used standard for communication in the power and energy industry. In particular, the IEC 61850-9-3, a profile of the PTP IEEE 1588 standard will be used in this thesis due to the wide compatibility with all kinds of hardware components.

It is, thus, the objective of this master's thesis to implement a clock synchronization concept for the existing PHIL testbed of FSU-CAPS. This work also demonstrates representing user applications to take advantage of the common clock synchronization.

1.1 Thesis Contributions

One major challenge of this thesis is to work with the already existing HIL testbed at CAPS, and the associated hardware requirements. Since the testbed is parallel used for different projects, the development process must not harm or interfere in any way with the ongoing simulations. This circumstance is called Brownfield development, compared to Greenfield development which is a totally new environment and allows development from a clean slate.

So the major thesis contributions can be settled as follows:

- Determining the appropriate Clock Synchronization Standard according to the existing hardware components.
- Classification of available devices, depending on the compatibility of clock synchronization.
- Selecting three representative devices for a demonstration test.
- Establishing a GPS synchronized Backbone with an external Grandmaster Clock (GMC).
- Validation of the system performance of the developed concept.

1.2 Description of the research content

The thesis can be separated into two major tasks. On the one hand, the PTP Backbone, which provides a GPS synchronized time reference with an interface to components of the CAPS HIL testbed. And on the other hand, the performance validation concept verifies the improvement and benefits on a scientific base. So the thesis can be structured further as follows. Chapter 2 contains background information about different concepts of clock distribution systems in HIL simulations. Followed by an overview of different PTPs, their benefits, use cases, and necessary background information. Chapter 2.4 treats the current HIL testbed of FSU-CAPS, including an explanation of used hardware components and workflow. Chapter 3 describes the practical part of the thesis. This covers the complete design process of the PTP Backbone. Chapter 4 copes with the performance validation process of the developed concept. This includes developed measurement concepts, the validation concept of the virtual network emulator, and the associated results. The last Chapter 5 provides a conclusion and a future outlook with further improvements and use cases.

1.3 Expected Outcome

The developed PTP Backbone should provide precise GPS synchronized time reference for devices that support the IEEE-1588 protocol or related profiles like IEC 61850-9-3. Components without such protocol support should also have the possibility of clock synchronization. One possible user application to demonstrate the advantages of common clock synchronization is the possibility of one-way path delay measurements. Since every component is synchronized to the same global time base, the path delay is the difference between hardware time stamped sent and received messages between two device. This provides higher accuracy compared to application-level Round Trip Time (RTT) measurements. In particular, if a system only implement software instead of hardware timestamp mechanisms. This gives a new opportunity to quantify the performance of CORE compared to real hardware. Moreover, since data of a HIL simulation has a timestamp from the common clock, data during post-processing can be compared with co-simulations.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

State of the Art

2.1 Precision Time Protocol IEEE 1588

The most common protocols in terms of clock synchronization are Network Time Protocol (NTP) and IEEE 1588, well known as PTP. NTP can provide microseconds time accuracy using software-based solutions, where PTP achieve up to tens of nanosecond using hardware timestamps. Typically, Global Navigation Satellite Systems (GNSS)/GPS are used as the main time source due to their price, global time base, accuracy, and easy deployment. Such systems allow the use of time (Coordinated Universal Time (UTC)) and frequency references with tens of nanosecond accuracy [12]. Another important aspect is cyber security, Lisova et al. [13] applied a game theory to investigate possible strategies to perform attacks on clock synchronization systems but also a network monitoring concept aiming to detect such attacks. The following section should give general information about the necessary PTP terms, to understand and configure a PTP synchronized network.

2.1.1 Clock Types

In PTP network four different types of clocks exist, **Ordinary Clocks**, **Grandmaster Clock**, **Boundary Clocks**, and **Transparent Clocks**. Ordinary clocks are typically devices with only one port, also called end nodes, and can be selected as GMC from the Best Master Clock Algorithm (BMCA) [14]. Only one GMC can be selected at the same time. Boundary Clock (BC)s are usually switches or routers in the PTP network and thus, have more than one PTP port. BCs synchronize their own clock with the selected GMC, an example is explained in Figure 2.1. Transparent Clock (TC)s can be separated into two categories, End-to-End (E2E)-TCs and Peer-to-Peer (P2P)-TCs. E2E-TCs are forwarding PTP synchronization messages but do not provide correction for the internal propagation delay of the link itself. P2P-TCs on the other side are considering this internal link delay. Figure 2.2 shows an example of master-slave clock synchronization with a P2P-TC in between [15].

2.1.2 Clock Synchronization

To synchronize all clocks in a network and thus, compensate for clock offset and clock drift, it is required to calculate the link delay between each device. Therefore, two synchronization concepts exist depending on the use of BC or P2P-TC. Figure 2.1 shows an example of the fundamental peer delay mechanism of IEEE 1588 using a BC. The master is sending a *Sync* message and a *Follow_up* message containing the hardware timestamp of sending the *Sync* message (t_1). The slave is hardware time stamping the receiving of the *Sync* message (t_2) and also saves t_1 from the *Follow_up* payload. The slave is sending a *Delay_Req* to the master and timestamps the sending time (t_3). The master is answering with a *Delay_Resp* message containing the timestamp of receiving the *Delay_Req* message (t_4) [1].

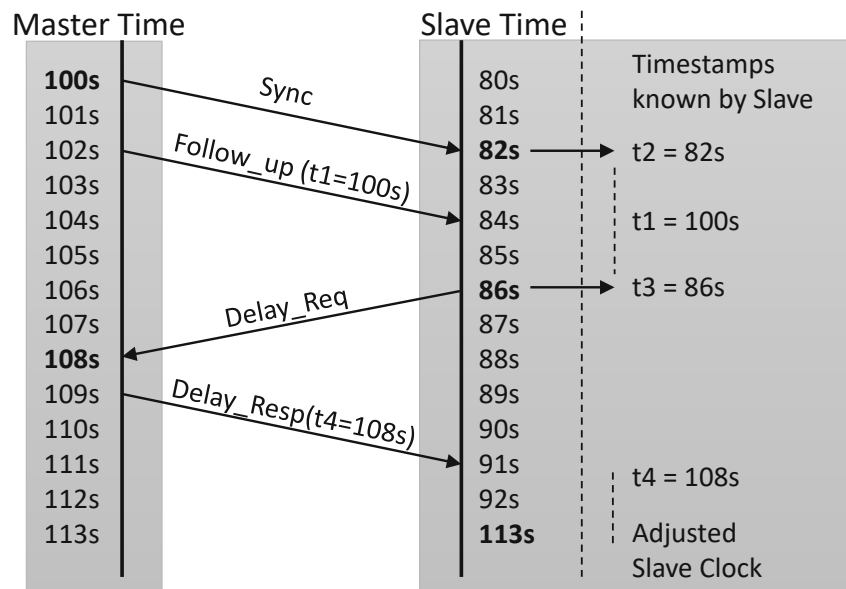


Figure 2.1: Example peer delay mechanism between master and slave clock [1].

The slave has now four timestamps (t_1 , t_2 , t_3 , t_4) locally available to calculate the current clock offset between master and slave according to Equation 2.2. In this particular example, the initially unknown link delay is 2s and thus, the clock offset between slave and master is -20 s. The *RateRatio* is an indicator of clock drifts, but the algorithm is not prescribed in IEEE 1588. Any algorithm is allowed that achieves a measurement accuracy of ± 0.1 ppm. Equation 2.3 represents an example algorithm introduced in [16]. Since the peer delay mechanism is repeated periodically, the rate ratio is calculated as the ratio of the arrival intervals of consecutive *Delay_Req* messages to the departure

intervals of the same message.

$$Clock_offset = RX(t_4) - TX(t_1) - Link_delay = 2\text{ s} \quad (2.1)$$

$$Link_delay = \frac{[(t_4 - t_3) + (t_2 - t_1)]}{2} = -20\text{ s} \quad (2.2)$$

$$RateRatio = \frac{t'_1 - t_1}{t'_2 - t_2} = \frac{105 - 100}{87 - 82} = 1 \quad (2.3)$$

Figure 2.2 shows an example P2P mechanism with a transparent clock switch between a master and a slave clock. Every Master clock is sending a *Sync* message to the network and TC/BCs are forwarding this *Sync* message to the slaves. Every node in this path is saving the time of sending and receiving locally. Then, a *Follow_up* message including the sending timestamp of the master *Sync* message is also sent to the slave over the same path as the *Sync* message. The announce message contains clock properties like clock accuracy, and a *timeTraceable* flag which indicates that the GM is using a GPS reference or other clock information. According to all announce messages in the network, the BMCA selects the GMC [14]. After that, the link delay between nodes can be calculated as analog to the explained BC mechanism in Figure 2.1. The time offset between the master and slave clock can be calculated with Equation 2.4. The correction field contains the time delay within the switch to forward a message.

$$Time_offset = RX(t_4) - TX(t_1) - \sum Delay - Corr_field \quad (2.4)$$

$$Delay_1 = \frac{[(t_6 - t_5) + (t_8 - t_7)]}{2} \quad (2.5)$$

$$Delay_2 = \frac{[(t_{10} - t_9) + (t_{12} - t_{11})]}{2} \quad (2.6)$$

$$Corr_field = (t_3 - t_2) \quad (2.7)$$

Important to mention is, that both synchronization methods are based on the assumption of symmetric link delay.

2.1.3 PTP Profile IEC 61850-9-3

The IEEE 1588 definition of a PTP profile is the set of allowed PTP features applicable to a device. A PTP profile is usually specific to a particular type of application or environment and defines the following values [15]:

- Best master clock algorithm options.
- Configuration management options.
- Path delay mechanisms.

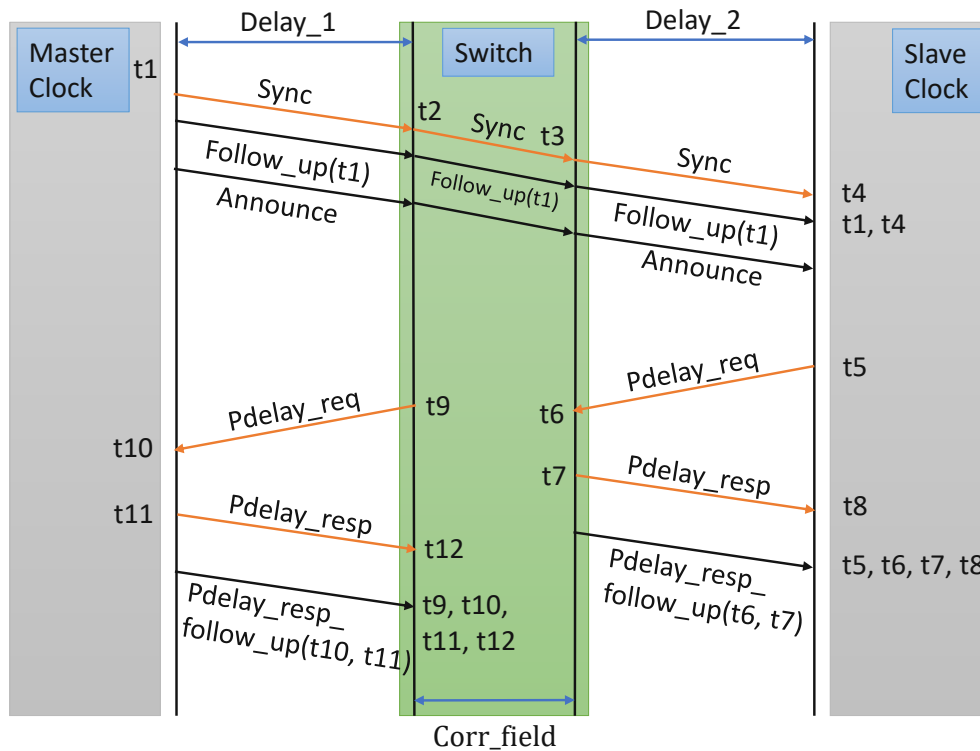


Figure 2.2: Peer-to-Peer mechanism with a transparent clock [2].

- Range and default values of all PTP configurable attributes and data set members.
- Transport mechanisms that are required, permitted or prohibited.
- Node types that are required, permitted or prohibited.
- Options that are required, permitted or prohibited.

Since 1960 when digital communication became more and more relevant different communication protocols were developed. But requirements from the past like reducing bytes on the wire are obsolete nowadays. Therefore, the International Electrotechnical Commission (IEC) has developed and released a new global standard for substation automation called IEC 61850. The main structure of that protocol is abstracting the data items and services. The so-created data is independent of any underlying protocol and can be mapped to any other protocol that can meet the specific requirements [17].

One part of that new standard is the IEC 61850-9-3, also called Power Utility Profile (PUP), clock synchronization and precise timing distribution protocol. PUP is a sub-profile of the IEEE 1588 PTP. IEC 61850-9-3 defines specific or allowed values for the PTP network used for power the substation. Power profile values provide a consistent

and reliable network time distribution system within and across substations [15].
Some Requirements for IEC-61850-9-3 PTP networks:

- GMC accuracy has to be smaller than 250 ns.
- All clocks have to send PTP messages over network protocol 802.3 (layer 2).
- All clocks have to support P2P mechanism.
- All clocks have to support 1 and 2 step synchronization.

Additional timing constraints are documented in [18].

2.2 Hardware in the Loop Simulations

HIL simulation is a real-time simulation technique that, when combined with emulated input and output signals, supports the development and testing of real-world systems. Real-time simulators typically abstract and model any complex system. At the same time, DUT, a specific component whose behavior is evaluated, is not modeled or simulated, but actually works as if it were installed in a real plant. In other words, the HIL simulation platform emulates the interaction between the plant and the DUT, usually by mimicking electrical input/output signals. Furthermore, simulating plant behavior in real-time using specific mathematical models. HIL provides and receives control signals and receives feedback from the DUT. This allows to repeat simulations for multiple runs under the same conditions and thus, greatly improves the reproducibility of the test process under development [5].

The DUT can be a sensor, an electrical machine, a generator or an entire system. They cover the whole scope two further categories exist, CHIL and PHIL. Figure 2.3 illustrates the basic structure of Controller and Power Hardware in the Loop simulations.

RTS typically use a fixed-time step solver with the smallest time-step size that can be achieved. This limitation can affect the time constants, and thus the switching frequencies of power electronics models. In addition, the amplifiers, actuators, sensors, and A/D cards of a PHIL simulation have their own bandwidth, which can cause time delays to create instability in some cases. The power limits of amplifiers and actuators (voltage, current, torque, speed, etc.) also add constraints that need to be fulfilled. Therefore, it is important to consider the limitations of PHIL experiments and properly assess the system requirements [3].

CHIL simulations compared to PHIL are typically handled with low voltage levels (± 10 V). Figure 2.4 illustrates the layer topology and a possible CHIL setup. Depending on the use case, a RTS can emulate a test system in the physical layer of the OSI model. Lower-level Controller hardware can be added for case-dependent control schemes (e.g. Frequency/voltage drop control). Control architecture, or cyber nodes, are the interface

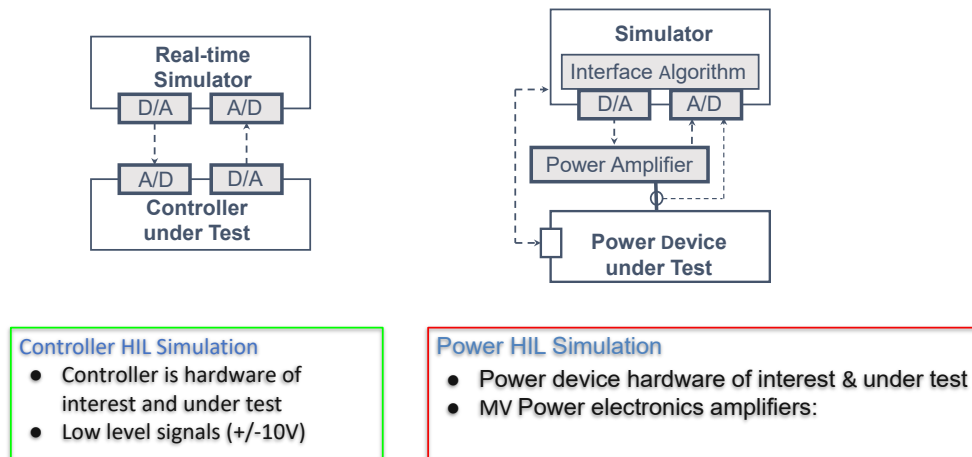


Figure 2.3: The Basic structure of PHIL/CHIL-based experimental test [3].

between physical layer and the cyber layer and typically communicates via Modbus Transmission Control Protocol (TCP)/Internet Protocol (IP) protocol. Another part of the cyber layer is the aggregator which acts as Human-Machine Interface (HMI) to forward information to the cyber nodes [4][19].

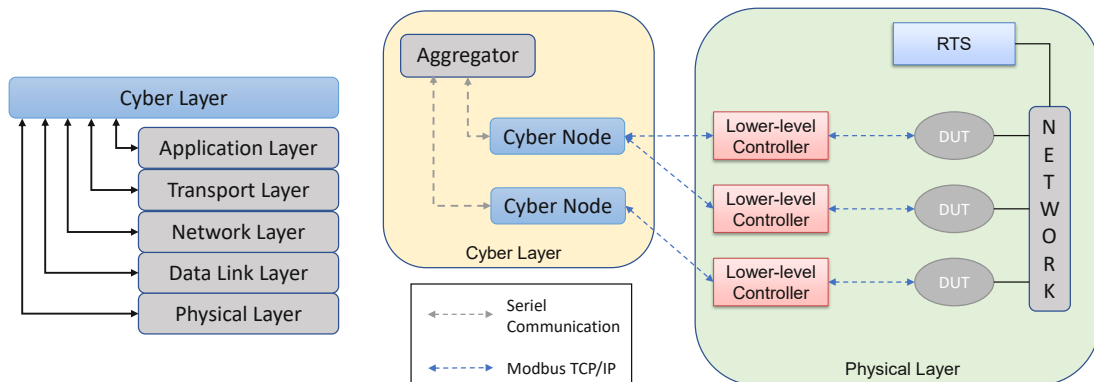


Figure 2.4: OSI layer topology and an illustrated CHIL setup [4].

2.3 Case Studies

2.3.1 GPS Clock Synchronization Accuracy

The position of a GPS receiver on earth can be calculated by multiplying the one Pulse per Second (1PPS) signal received from a satellite with the speed of light. With three satellites the current position (x, y, z) can be determined. Since the internal clock of GPS receivers has a certain accuracy, a clock drift could be interpreted as a movement

of the receiver or clock drift. Therefore, a fourth satellite is required to verify the time error of GPS receiver's clock. Since the receiver is measuring the distance to satellites at the same time, each measurement includes the same time offset error. Therefore, calculated ranges between satellite and receiver with the same measurement error are called pseudoranges [20].

The work from Lee et al. [21] demonstrates a concept to calculate the time difference between two distributed GPS synchronized ground stations. One receiver is in Korea Standards Research Institute (KSRI) and the other in Tokyo Communications Research Laboratory (CRL). The distance between both is ~ 3000 km. Each GPS receiver is down-converting the 1pps signal from the antenna to a Intermediate Frequency (IF) of 75.42 and 10.7 MHz. The GPS receiver gets a 1PPS signal from the antenna and calculates its clock offset compared to the satellite's according to Equation 2.8

$$\Delta T_{is} = t_i - t_s = T_i - \rho_i/c - \delta\rho_i/c - t_{ri} \quad (2.8)$$

where

T_i	... pseudorange time interval.
t_i	... 1PPS signal of clock i.
t_s	... 1PPS signal of satellite.
ρ_i	... geometric delay between a station and a satellite.
$\delta\rho_i$... error sources.
t_{ri}	... receiver delay time.
c	... speed of light.

Main error sources are, geometric delay, ionospheric, tropospheric and Sagnac effect [22]. Further, the clock time offset between two ground stations A and B can be calculated by:

$$\tau_{AB} = \Delta T_{AS} - \Delta T_{BS} \quad (2.9)$$

$$\text{where } \Delta T_{AS} = t_A - t_s, \quad \Delta T_{BS} = t_B - t_s \quad (2.10)$$

During a four-day measurement period and a connection to only three satellites, the peak variation between both ground stations was 30 ns. The frequency stability between the receiver clocks of KSRI and CRL stations was about 1 part in 10^{13} [21].

Another approach to testing GPS clock synchronization accuracy demonstrates Jefferson et al. [23]. They are using reference data from 24 GPS satellites and more than 150 ground stations. The results presented in this work, are from stations with TurboRogue receivers in combination with H-Maser oscillators. Data acquisition is processed with the software tool GIPSY-OASIS II from the Jet Propulsion Laboratory (JPL) [24]. This software tool can measure and estimate GPS satellite clocks and orbits, receiver clock offsets and locations, and other parameters like Earth orientation. They present the clock accuracy between four ground stations, the smallest and highest offsets are highlighted here. Table 2.1 shows the results of clock accuracy between Tidbinbilla-Australia (TID2),

Algonquin Park-Ontario Canada (ALGO), and Pietown-New Mexico USA (PIE1). Due to the GIPSY-OASIS II tool and the estimation strategy, clock calibration between distributed ground stations of sub-one-nanoseconds can be achieved.

Table 2.1: 72-hour H-Maser linear fit residuals.

	TID2 vs ALGO	PIE vs ALGO
clock-offset	-43.32 ns/day	-16.24 ns/day
clock lin. fit	rms 0.411 ns/3days	rms 0.244 ns/3days
distance	16 000 km	3000 km

2.3.2 Clock Synchronization in HIL Systems

Rinaldi et al.[5] presented a method to extend a PHIL testbed with a PTP synchronization concept with minimal hardware requirements. Since some devices do not support the IEEE 1588 PTP they designed a software-based solution to timestamp analog signals during the simulation. To fulfill the timing requirements of the power industry, the power profile of IEEE 1588 (a.k.a. IEEE C37.238-2017) has been used.

The PTP Backbone which provides and distributes the precise GMC consists of three Siemens Ruggedcom RSG2488 switches and a GPS receiver. The Siemens switches fully support PTP/GPS/NTP and 1 Gbps fiber optic ports. One switch is connected to the GPS receiver and acts as GMC in the PTP Backbone. Another switch is used as an interface to IEEE 1588 DUTs and the third is used as an interface to the PHIL. The distances between those substations are between 300 m and 500 m. Figure 2.5 illustrates the explained PTP Backbone and the PHIL system.

The PHIL system is based on an OPAL-RT OP5700 RTS, operating on a Virtex-7 Field Programmable Gate Array (FPGA) platform, a Redhat OS, IntelXeon E5, 8 Cores, 3.2 GHz, and 8 GB RAM. In addition, four quadrants power amplifier, electronic load, an OP5330 analog output board, and an OP5340 input board with 1 MSamples/s.

Some devices of the PHIL supply part are high-performance analog input and output devices that cant be connected to the PTP Backbone. Therefore, a software-based solution is designed, see Figure 2.6. This approach follows the principle of a Phase-Locked-Loop. The PHIL system has a tunable internal clock *PHIL_Clk* which generates 1PPS signals with the Puls Generator. The Digital Analog Converter (DAC) block is looped back as input for the Analog Digital Converter (ADC) Block. An external Ref (UTC) signal and *PHIL_gen* 1PPS signal will be compared in terms of frequency and phase difference. PI Controller tunes the Pulse Generator until both signals have the same phase and frequency. Then the system is in a lock state and the internal 1PPS is synchronized to the external reference signal. If this external reference signal is provided by an IEEE 1588 device the PHIL is locked to the same common time base.

Figure 2.7 can be used to compare the generated 1PPS PHIL output with the IEEE 1588 1PPS. Therefore, a Keysight 53230A frequency counter is used to perform the clock

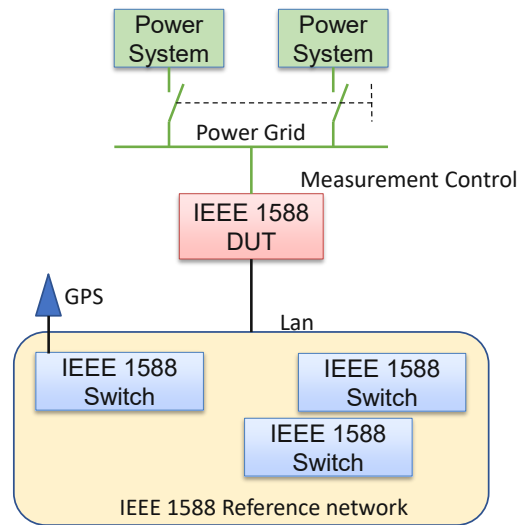


Figure 2.5: A concept overview to integrate IEEE 1588 power profile into a PHIL testbed [5].

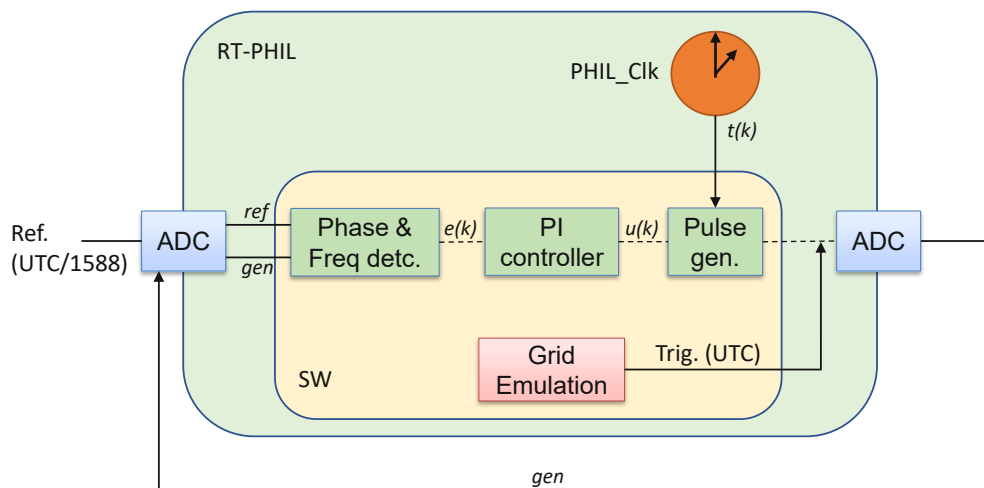


Figure 2.6: The proposed software-based time synchronization mechanism [5].

stability measurement by measuring the frequency of both signals. The Siemens IEEE 1588 switches do not provide a 1PPS signal. Therefore, a APU2C4 board with an AMD G series GX-412TC CPU, 1 1 GHz quad-core, 64 bit, 4 GB RAM with a Intel i210 NIC is used. The system runs on an Arch Linux distribution, kernel version 4.7.2. With the Linuxptp package and the ptp4l command, the Intel i210 can be configured as PTP slave and also provide 1PPS synchronized to the IEEE 1588 GMC.

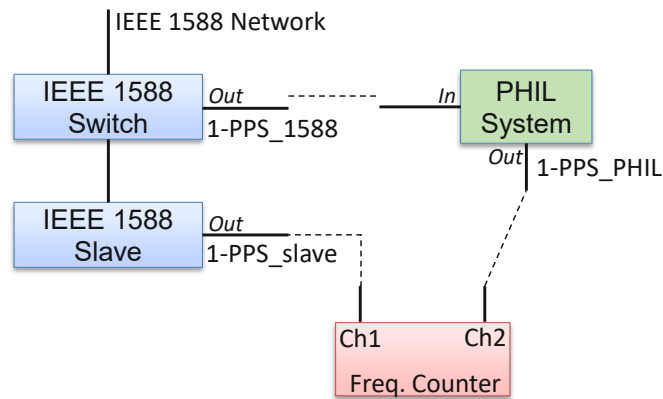


Figure 2.7: The block diagram of the experimental setup [5].

With the explained measurement method the percentage frequency distribution between the generated 1-PPS_{Phil} and 1-PPS_{1588} can be evaluated. Measured 43200 samples during 12 hours, leads to a mean time offset of $0.4\ \mu\text{s}$, a standard deviation of $0.25\ \mu\text{s}$ and jitter of $2.3\ \mu\text{s}$.

2.3.3 Common Open Research Emulator (CORE)

Ogilvie et al.[6] are investigating the communication network components of a Controls Evaluation Framework (CEF) for future shipboard platforms. The idea is to augment the CEF's available feature set with the help of CHIL simulations. Thus, the communication network components, the DUT, can be tested in a highly controllable environment. Co-simulations with multiple controllers, allow simulations with varying timesteps and different control schemes to simulate the required environment according to the specific use case requirements.

One part of the work is to replicate the in [6, Section 3 A] introduced hardware-based communication network with a Common Open Research Emulator (CORE) [25]. CORE is a software solution to emulate computer networks running on a high-performance server including multiple NICs. CORE is capable of emulating Ethernet switches/hubs, routers, wireless local area networks (WLANs), and virtual PCs. In addition, CORE can emulate applications, and the behavior of data link and physical layers of the OSI model [26]. The capability of running CORE simulations in real-time is the reason to be used as the software platform for the CEF communication network module.

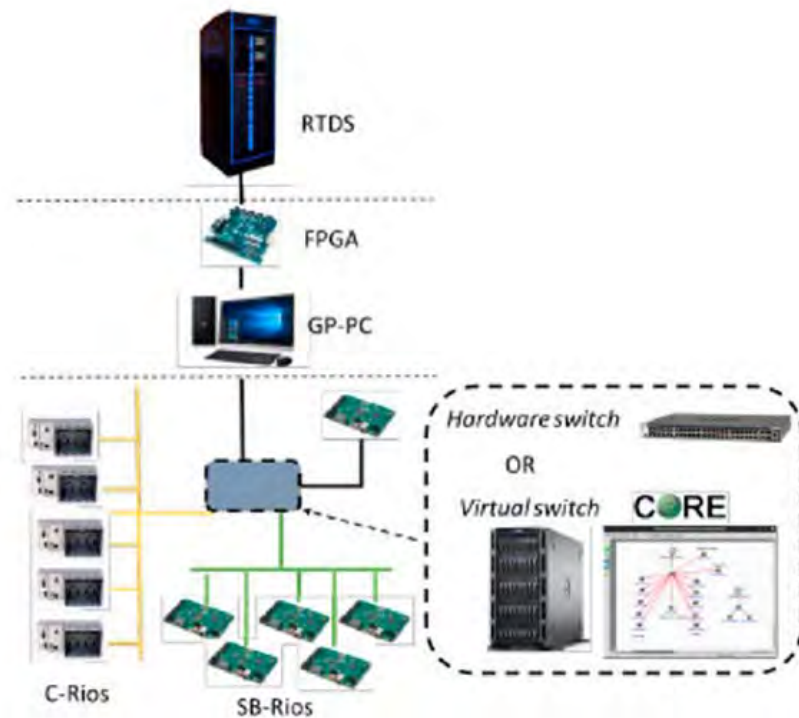


Figure 2.8: Overview of a representative CHIL setup to evaluate distributed energy management control system performance [6].

Figure 2.8 shows the overview of hardware components of the experimental setup for the distributed energy management control system performance validation. With RTDS as RTS system and CORE as a virtual network between C-Rios and SB-Rios controller. To evaluate the performance of an emulated network in CORE compared to a real network, the response time of a real hardware switch is compared with an emulated switch in CORE. For the experiment, a Netgear M4300-52G switch is used and CORE is running on a Dell server with Intel Xeon E5-2637 v4 CPU with 3.5 GHz, 64 GB RAM, and 26 i350 Gigabit network connections.

To measure the response time, a UDP client-server *echo* benchmark test similar to a *ping* command is performed. According to the measurements, the virtual switch results in a mean response time of $124 \mu\text{s}$ (see Figure 2.9b) and the real hardware switch $153 \mu\text{s}$ (see Figure 2.9a). Based on that measurement further response measurements are processed including the distributed energy storage module controllers (see [6, Section 4 B]). It is important to mention that the measured response time includes the latency for packet processing in the operating system and is not restricted to propagation time through the computer network. It is a typical system-level control that is likely performed by the operating system. Therefore, the response time can be considered representative.

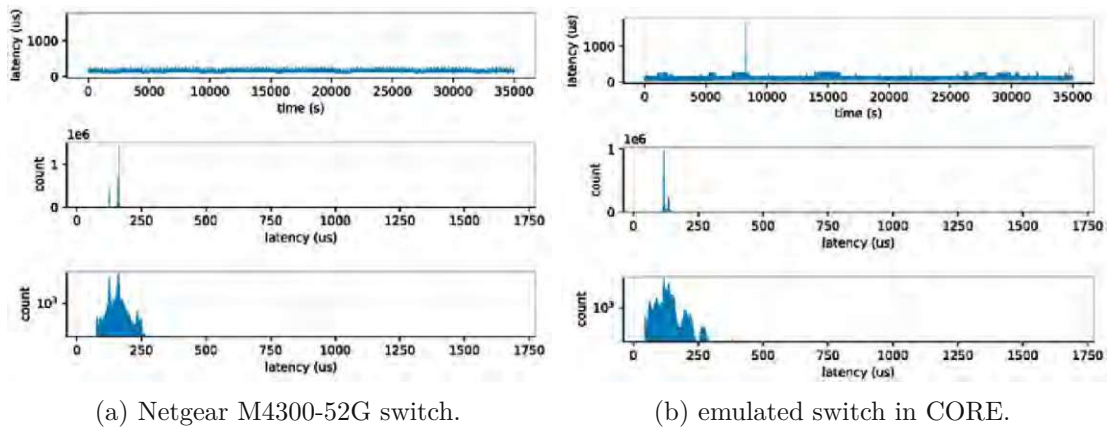


Figure 2.9: Response time between a) a real hardware switch and b) an emulated switch within CORE [6]

2.4 Current Hardware in the Loop Testbed

2.4.1 Center for Advanced Power Systems

CAPS at FSU is a leader in the field of PHIL simulations where CHIL is a fundamental aspect of the whole process. The following HIL setup which will be illustrated in this Thesis is not a specific instance of any prior or current experiment at FSU-CAPS but is representative of many HIL experiments [27][28][29][30].

2.4.2 Power Hardware in the Loop Testbed

Any HIL experiment will generally include one or more RTS. The RTS generally provide the following functions: real-time electrical stimulation, system-level controls/protections, and HMI. In terms of real-time simulation, "real-time" refers to the deadlines associated with synchronizing the simulated system with the physical world or wall clock time. By far the most common real-time model is a strictly periodic task model that computes simulated values and has a deadline corresponding to its period. The period of such a task is usually referred to as a simulation time step. FSU-CAPS uses simulators from most major RTS vendors, more specifically RTDS, OPAL-RT, Typhoon, and Speedgoat. For this work, RTDS is selected as represented RTS since it is used in most of the simulations at CAPS. The GTSync is an RTDS-specific card that allows a real-time simulation to be synchronized with an external clock and supports 1 Pulse Per Second (1PPS) or IEEE 1588 via RJ45 or ST fiber-optic connection. However, IEEE 1588 is currently not used. Instead, events that occur during a timestep period of RTDS are captured together with the current timestep counter since simulation start. This kind of time stamping is provided by the custom logger (GLogger) and data are stored in a Windows file server, in Figure 2.10 shown as "plasma".

The GLogger (GTFPGA Logger) is a data acquisition tool developed by FSU-CAPS

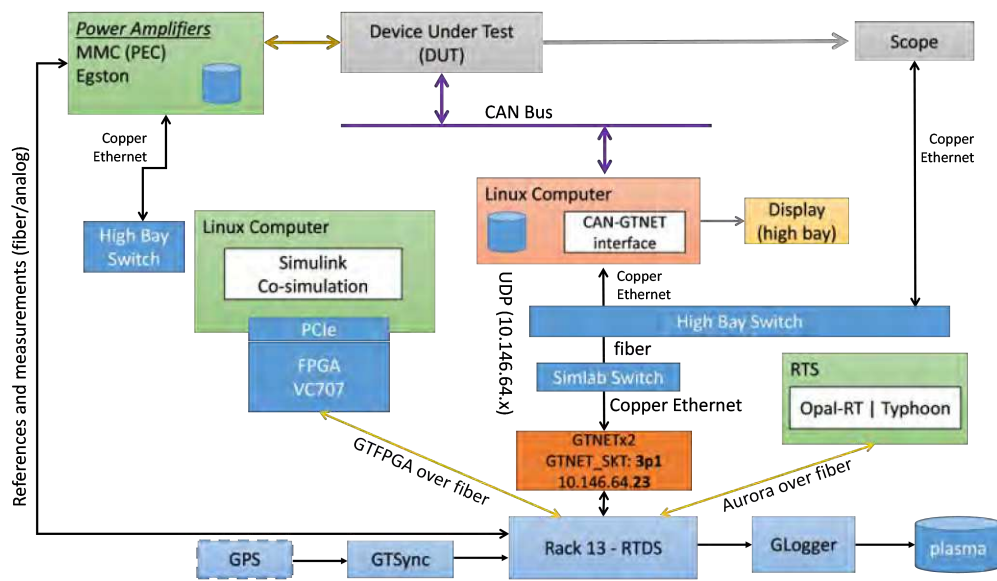


Figure 2.10: Current Power Hardware in the Loop Testbed setup overview.

and specifically designed for connection to RTDS racks and chassis. Communication is point-to-point using an RTDS proprietary 8b/10b encoded data stream. At each time step, the RTDS transmits values for a user-defined/configurable set of signals. The current method of time stamping is done by appending an integer denoting the values of the simulation time step. The start of an RTDS case/simulation begins at time step zero and the first time step values available to the GLogger are timestamped zero and incremented by one for each subsequent time step. The typical time step length is 25-50 ms.

Many DUTs communicate via the CAN protocol. Since RTDS is used for protection, monitoring, and as an HMI, data produced by a DUT is sent to RTDS. The data received from or sent to the DUT is recorded locally on the Linux PC and is timestamped with the time-of-day, typically synchronized with ntpdate software tool. Ntpdate sets the local date and time by polling the NTP server(s) to determine the correct time. The communication through the Linux PC (CAN-GTNET Interface) is best effort and only coarsely time synchronized with the values produced by RTDS (a back-of-the-envelope approximation is tens of msecs).

Scopes are best effort synchronized but could also benefit from global synchronized timestamps. Typical use cases are sporadically short duration (typically seconds long) captures at high resolution throughout a given experiment.

PHIL Amplifiers:

Parts of the Testbed is builds on a combination of megawatt-level power amplifiers such as the Variable Voltage Sources (VVS) and Multilevel Converters (MMC) built by ABB, and a 100 kW-level Egston amplifier as shown in Figure 2.11. The VVS power amplifier

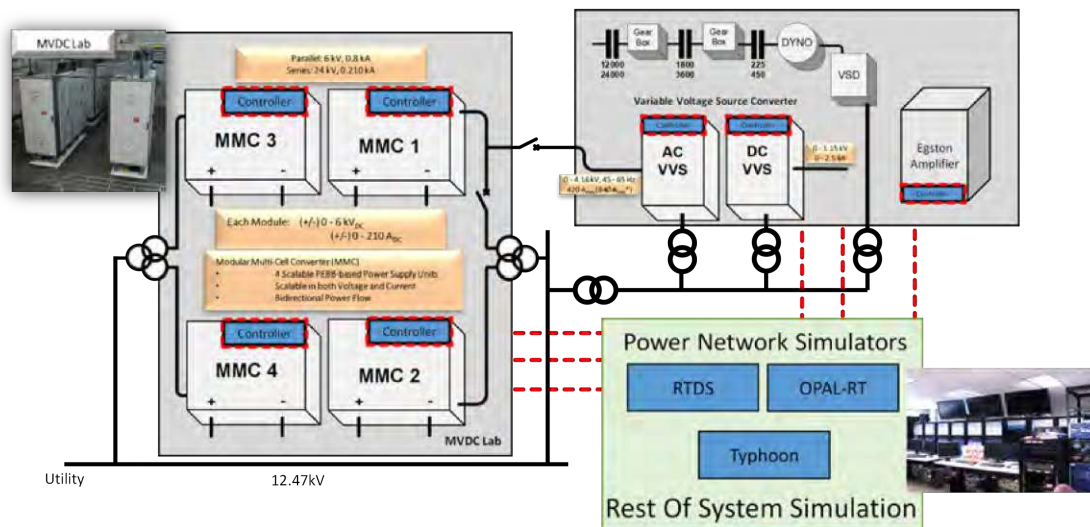


Figure 2.11: FSU-CAPS Power Hardware in the loop Amplifiers setup.

can be configured to operate in AC-only mode (5 MW, 0-4.16 kV, 45-65 Hz) and AC-DC mixed-mode with 2.5 MW each and dc-voltage of up to 1.15 kV. Reference signals are provided as analog signals to the VVS.

The MMCs provide direct current and are rated 6 kV and 208 A with full four quadrant capability. Combining the MMCs in series or parallel allows up to 24 kV and 832 A. Two MMCs can also be combined to operate back-to-back. Custom control modes allow use of the MMCs as inverters to provide AC voltages/currents. Reference signals are provided as analog signals.

The Egston amplifier at CAPS is a COMPISO System Unit (CSU) 100 kVA. It is based on six individually controllable single-phase amplifier modules with a frequency range from DC to 5 kHz full output voltage range. The reference values can be provided through a multi-mode fiber interface through an RTDS GPES interface box.

Each mentioned power system provides custom fault recording options. But, due to the lack of time synchronization, these features are currently not commonly used in experiments.

2.4.3 Controller Hardware in the Loop Testbed

Chapter 2.2 introduced the fundamental concept of HIL and also presented a basic CHIL setup. The CHIL testbed at CAPS uses the same infrastructure as the presented PHIL testbed for real-time simulations and monitoring. Since the DUT (e.g. a controller) requires low voltage input and output signals (typically ± 10 V) the power amplifier infrastructure introduced in the previous sections are not in use. Figure 2.4 from Chapter 2.2 shows that the test case implemented in the physical layer also includes a computer network. Depending on the specific use case, such networks can be complex

and thus hardware and cost intensive. To follow the fundamental idea of HIL simulations a virtual network emulator CORE comes into place to emulate the desired computer network around the DUT.

CORE is running in an appropriate high-performance Dell Server, with a Intel(R) Xeon(R) CPU E5-2637 v4 @3.5 GHz processor, 64 GB RAM and 26 Intel i350 network cards which can be used as the hardware interface between the real network and emulated network. Possible communications protocols between DUT and the aggregator are Modbus TCP/IP, Controller Area Network (CAN)-Bus, Aurora over fiber or Ethernet. A specific use case of CORE is explained in Chapter 2.3.3 and in a related work [11].

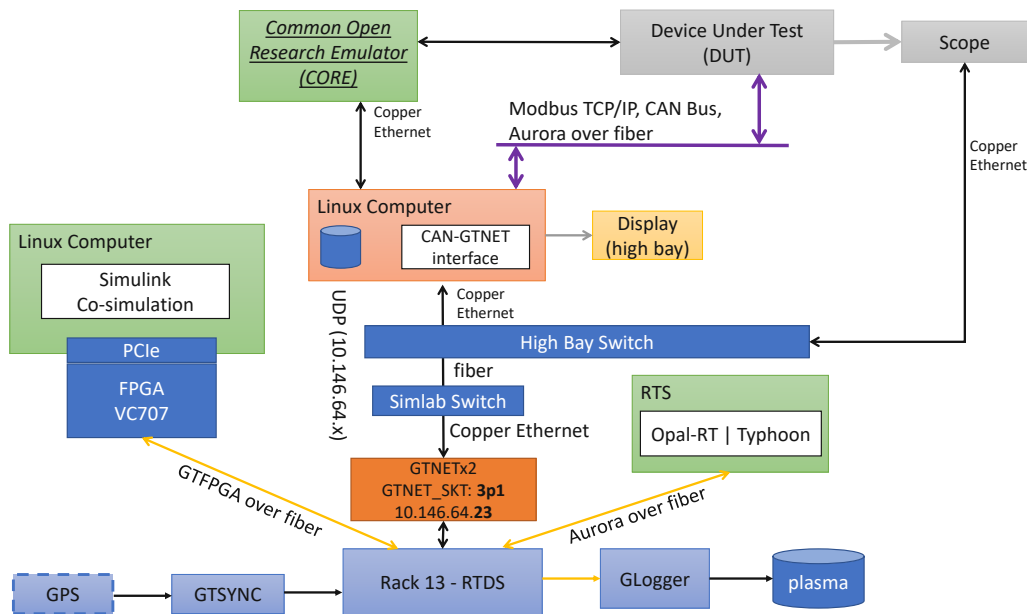


Figure 2.12: FSU-CAPS Controller Hardware in the loop setup.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

System Architecture / Implementation

To implement a clock distribution system in the current HIL testbed of FSU-CAPS, which was introduced in Chapter 2.4, it is necessary to classify the major components about their compatibility with mentioned PTP concepts from Chapter 2.1. The common PTP which is supported by each representative component is the IEC 61850-9-3. For an overall understanding Figure 3.1 illustrates the current HIL testbed with the extended PTP backbone (yellow area) developed during this thesis.

The GPS Receiver is the main time source of the whole PTP network, also called GM. It receives 1PPS signals from external GPS Antenna to provide the PTP to the whole network. An appropriate PTP Switch is necessary to distribute the precise GMC. A 10 MHz reference signal which is synchronized to the PTP network, can provide a precise time reference for devices without PTP support. The following sections will give a detailed description of the development, including all shown components, their functionalities, and configurations.

3.1 PTP Compatibility of HIL Interfaces

According to the research of commonly used PTPs in Chapter 2, IEEE 1588 is widely used and supported by components in the data acquisition sector. Nevertheless, since IEEE 1588 has different sub-profiles like IEC 61850-9-3, it's necessary to specify the compatibility of major used hardware components in the HIL testbed.

3.1.1 Real Time Digital Simulator

Since RTSs from RTDS vendor are the most commonly used simulator hardware at CAPS, the focus on finding an interface to a PTP network was on RTDS components. RTDS

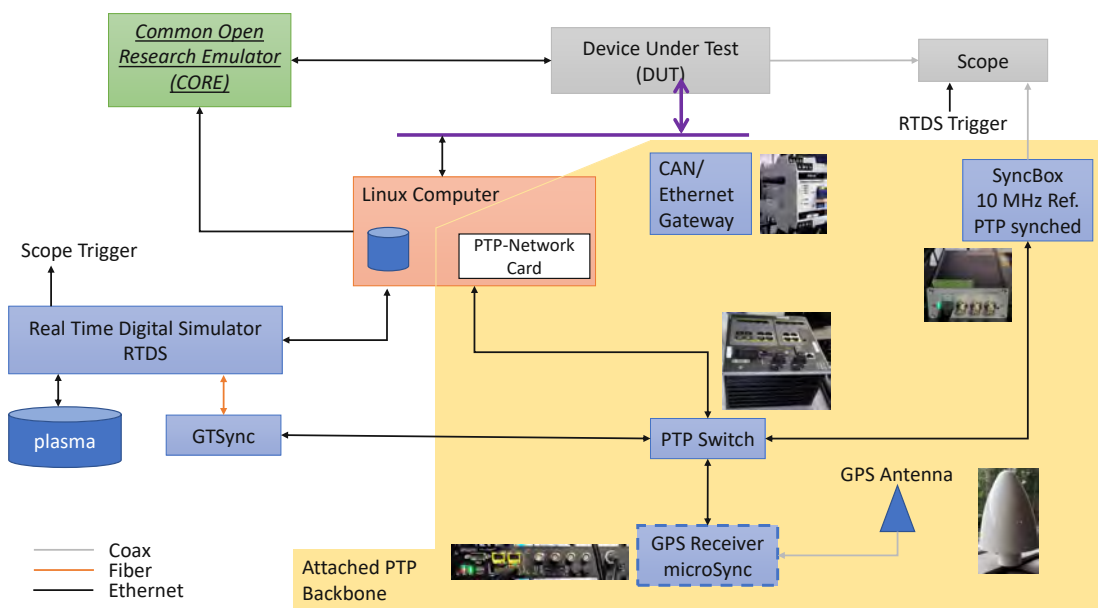


Figure 3.1: Extended CHIL testbed with a PTP backbone.

uses three different generations of main processing units (GPC, PB5, and NovaCor) for the real-time simulation, implemented in FPGAs. The GTWIF card is the interface between GPC and external networks, the Graphical User Interface (GUI) RSCAD or any other RTDS card. So any configuration of used cards is accessible via the GTWIF terminal [31].

To synchronize simulation timestamps, RTDS offers a specific hardware component called GTSYNC card. It can use its own internal time source or external references like IRIG-B, 1PPS, IEEE 1588, IEEE C37.238 and IEC 61850-9-3. Table 3.1 gives an overview of the available IEEE 1588 profiles and their supported features. The only network protocol in combination with PTP that is supported from the GTSYNC card is IEEE 802.3 (Layer 3), P2P delay mechanism and acceptable data rate is limited to 100 Mbps[7].

Table 3.1: Overview of PTP features of a GTSYNC card.

Delay Mechanism	Network Transport	PTP Profiles	Data Rate
Peer to Peer	IEEE 802.3 (Layer 2)	IEEE 1588 IEC 61850-9-3 IEEE C37.238	100 Mbps

Figure 3.2 illustrates how the mentioned components are connected within a RTDS rack. The GTNET card is a real-time communication interface between the simulation and an external network. It has a fiber connection to the GPC/NovaCor card and implements various types of communication modules. One of them is the GTNET-SKT for UDP communication protocol, which is necessary for the performance validation, which will be

explained in Chapter 4. The Inter Rack Global Bus Hub (IRGBH) manages inter rack time synchronization and is the interface between GTWIF cards of different racks. This connection is relevant during parallel simulations across multiple RTDS systems.

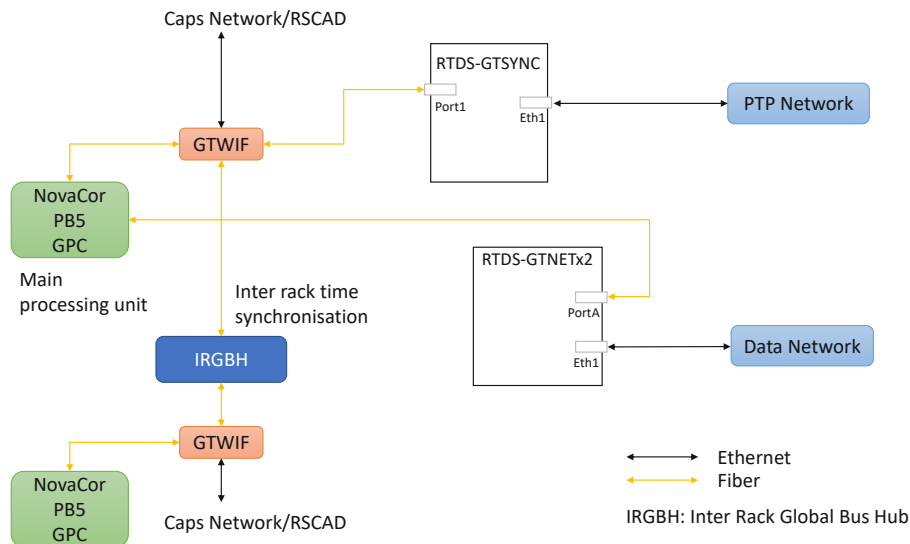


Figure 3.2: RTDS periphery including different interface cards [7].

The connectors and periphery of a GTSYNC card is available in the RTDS GTSYNC manual [7]. Besides the 24 V power supply three further interfaces were used. The 100base-TX Copper Ethernet Port is connected to the PTP Backbone. The GT Port 1 is connected to the GTWIF card via fiber. And the several status led around the board for visual monitoring [7].

3.1.2 Linux Operating System

Since computers with Linux Operating System (OS) are a major component in every HIL simulation, it adds a significant benefit if their internal clocks are synchronized to the common time base. Therefore to include a Linux machine to the PTP Backbone, specific software and hardware requirements must be met. Two major PTP implementations to synchronize to IEEE 1588 are mentioned in the literature, PTP daemon (PTPd) [32] and Linuxptp [33]. According to the PTPd manpage [34] and Wiesner and Kovacszy [33, Section 1], hardware timestamps are not supported but considered in future updates. Since the PTPd GitHub repository [35] is inactive over the last few years, this software solution is not promising.

Linuxptp on the other hand, supports hardware time stamping and is widely used in the community. This implementation has well-known issues (e.g., see Machnikowsk et

al. [36]), but has an active community that continuously develops and commits updates to the Linuxptp GitHub repository [8]. Table 3.2 gives a short overview of important parameters that are relevant for GMC and PTP switch selection.

Table 3.2: Overview of PTP features of Linuxptp.

Delay Mechanism	Network Transport	PTP Profiles	Time Stamping
Peer to Peer	IEEE 802.3 (Layer 2)	IEEE 1588	Hardware
End to End	UDP IPV4 (Layer 3)	IEC 61850-9-3 IEEE C37.238	Software

A NIC is the first instance of receiving data from the network, time stamping the moment of receiving the data packet before further processing. To achieve nanosecond accuracy of such timestamps a NIC needs to support hardware time stamping. Table 3.3 shows an example of a NIC that meets requirements for hardware timestamps. This can be verified with the Linux shell command:

```
$ethtool -T eth6
```

Table 3.3: Hardware requirements for a NIC to communicate with IEEE 1588 PTP [8]

Time stamping parameters for: eth6

Capabilities:

hardware-transmit	(SOF_TIMESTAMPING_TX_HARDWARE)
software-transmit	(SOF_TIMESTAMPING_TX_SOFTWARE)
hardware-receive	(SOF_TIMESTAMPING_RX_HARDWARE)
software-receive	(SOF_TIMESTAMPING_RX_SOFTWARE)
software-system-clock	(SOF_TIMESTAMPING_SOFTWARE)
hardware-raw-clock	(SOF_TIMESTAMPING_RAW_HARDWARE)

PTP Hardware Clock: 0

Hardware Transmit Timestamp Modes:

off	(HWTSTAMP_TX_OFF)
on	(HWTSTAMP_TX_ON)

Hardware Receive Filter Modes:

none	(HWTSTAMP_FILTER_NONE)
all	(HWTSTAMP_FILTER_ALL)

The NIC I350 (2 or 4 port) from Intel fulfills the mentioned functionalities from Table 3.3, and each port supports 10/100/1000 Mbps data rates. Since this NIC was available at CAPS, it is used in every setup where a Linux PC is synchronized to the PTP network.

3.1.3 Non PTP Compatible Devices

Not every hardware component in the field can communicate with a PTP like IEEE 1588, therefore other solutions for the best effort synchronization have to be made. Scopes are widely used in different kinds of setups for a short snapshot of the DUT output or

long-term monitoring. Scopes like Tektronix MDO4034C/MSO58-2-125 are frequently used during HIL simulations at CAPS. Each device has an internal oscillator as a time reference. The stability of such oscillators are highly dependent on their quality, age, and changes in temperature. Therefore to guarantee precise long-term measurements, without clock drifts, a more precise external reference is necessary. For this purpose, scopes usually have a 10 MHz external reference input (BNC connector). According to the datasheet, the Tektronix MSO58-2-125 can phase-lock to an external reference with the precision of ± 4 ppm [37]. So a device is necessary which supports the IEEE 1588 PTP and provides a 10 MHz output that is synchronized to the PTP network.

The SyncBox/N2X [38] from Meinberg is an optimal solution for this purpose. It has three Programmable Pulse Output (PPO) (BNC connector) like 10 MHz, 1PPS, a 1 to 10 MHz frequency synthesizer, Inter-Range Instrumentation Group (IRIG) time codes and serial time telegrams. Some functionalities are not relevant for this thesis, but offers a huge variety for future use cases. The SyncBox has a built-in TCXO oscillator. The detailed specification is available in Table 3.4.

Table 3.4: Oscillators integrated in Meinberg Devices [9]

	TCXO	OCXO MQ
short term stability, ($\tau = 1$ sec)	$2 \cdot 10^{-9}$	$2 \cdot 10^{-10}$
accuracy of PPS, (pulse per sec)	$< \pm 100$ nsec	$< \pm 50$ nsec
phase noise	1 Hz -60 dBc/Hz 10 Hz -90 dBc/Hz 100 Hz -120 dBc/Hz 1 kHz -130 dBc/Hz	1 Hz -75 dBc/Hz 10 Hz -110 dBc/Hz 100 Hz -130 dBc/Hz 1 kHz -140 dBc/Hz
accuracy	$\pm 1 \cdot 10^{-7}$	$\pm 1.5 \cdot 10^{-9}$
free run, one day	± 1 Hz (1)	± 15 mHz (1)
accuracy GPS-synchronous, averaged 24 h	$\pm 1 \cdot 10^{-11}$	$\pm 5 \cdot 10^{-12}$
accuracy of time, free run, one day	± 4.3 msec	± 65 μ s
temperature dependant drift free run	$\pm 1 \cdot 10^{-6}$ ($-20 \dots 70$ °C)	$\pm 5 \cdot 10^{-8}$ ($-20 \dots 70$ °C)

Note 1:

The accuracy in Hertz is based on the standard frequency of 10 MHz.

For example: Accuracy of TCXO (free run one day) is $\pm 1 \cdot 10^{-7} \cdot 10$ MHz = ± 1 Hz

The 10 MHz output is based on a Transistor-Transistor Logic (TTL) and provides a 0 to 5 V rectangular signal. The input port (RJ45 Ethernet) accepts NTP and PTP IEEE 1588, UDP/IPv4 (L3) or IEEE 802.3 (L2) network protocol and E2E/P2P delay mechanism but only supports data rates of 10/100 Mbps. The SyncBox supports Power over Ethernet (PoE) as a power supply or 20 to 60 V DC via a power connector on the back of the device. For convenience reasons, PoE is preferred but adds an additional requirement for the PTP switch. To verify the accuracy of the 10 MHz reference, a second SyncBox is installed in the system to compare possible variations between them. This can be important since multiple SynBoxes could be placed in different locations of the

same HIL simulation. Therefore the deviation between 10 MHz references signals from the SyncBoxes should be known well.

3.2 PTP Backbone

Since the PTP Backbone developed during this thesis will be used from FSU-CAPS for current but also future HIL simulation, the system should provide maximum flexibility. Therefore Table 3.5 shows all mandatory and optional requirements according to occurring PTPs in the literature and the limitations of components of the testbed. Nevertheless, the mandatory features that should be supported from the GMC and PTP switch are: supporting network protocol IEEE 802.3 and P2P, and implementing the PTPs IEEE 1588 and IEC61850-9-3. The SyncBox and the GTSYNC card do not support data rates faster than 100 Mbps. Therefore, several SFP-GB-GE-T, 10/100/1000 BASE-T, RJ-45 Small Form-factor Pluggable (SFP) transceivers are used in the field. Since the time window for completing the PTP Backbone at CAPS was limited to three months, additional timing constraints like delivery time of needed components had to be considered as well.

Table 3.5: Mandatory (M) and optional (O) requirements for the PTP Backbone

Delay Mech.	Network Transport	PTP Profiles	Data Rate
P2P (M)	IEEE 802.3 (M)	IEEE 1588 (M)	10/100/1000 Mbps (M)
E2E (O)	IPv4/v6 (O)	IEC 61850-9-3 (M)	
		IEEE 802.1AS (O)	

3.2.1 Grandmaster Clock

From different vendors like Orolia, Oscilloquartz, Omicron, the Meinberg microSync RX102 GM in combination with the Meinberg SyncBox fulfill all technical and organizational requirements mentioned so far. Table 3.6 shows that the Meinberg GM offers a huge variety of supported network and time protocols in addition to the mandatory IEEE 802.3 and IEC 61850-9-3 protocols. All four SFP ports of the microSync RX102 support 10/100/1000 Mbps and two of them can be used as PTP reference for two different networks. To ensure all transmission rates, four SFP-GB-GE-T SFP transceivers from fs.com are used. The two PPOs of the microSync RX102 provide similar functionalities as the SyncBox. But in terms of local circumstances of the HIL testbed, DUT and GM are usually not at the same location. But scopes have to be placed right next to the DUT. So the big advantage of the SyncBox is, that the GMC can be distributed over the Ethernet PTP Backbone over longer distances. And only short coax cables are necessary for the 10 MHz reference for the scope.

Meinberg offers three different GPS antennas in combination with the microSync RX102 which can be seen in Table 3.7. Since the PTP Backbone has no redundant parallel GPS receivers where the BMCA can choose another time source in case of connection losses, the antenna should be as reliable as possible. Therefore the GNS-UC antenna

Table 3.6: Protocols and profiles of the microSync RX102 GM [10]

Network Protocols	IEEE 1588 PROFILES}
IPv4, IPv6	IEEE 1588v2 Default Profile
NTPv3, NTPv4	IEEE C.37.238-2011 Power Profile
PTPv1, PTPv2	IEEE C.37.238-2017 Power Profile
IEC 62439-3 (PRP)	IEC/IEEE 61850-9-3 Power Utility Profile
DHCP, DHCPv6	Enterprise-Profile
DSCP	ITU-T G.8265.1, ITU-T G.8275.1, ITU-T G.8275.2 Telecom Profile
IEEE 802.1q VLAN filtering/tagging	SMPTE ST 2059-2 Broadcast Profile
IEEE 802.1p QOS	IEEE 802.1AS TSN/AVB Profile
SNMPv1/v2/v3	AES67 Media Profile
Remote Syslog Support (UDP)	DOCSIS 3.1

is selected as the GPS source. This GNS-UC antenna establishes a connection to two satellite systems, GPS and Galileo. Thereby, it's very unlikely that both systems would fail and causes the GM to switch into Holdover mode. In this case, the OCXO MQ Oscillator within the Meinberg receiver could provide a time source with an accuracy of $\pm 65 \mu\text{s}$ over a period of one day and $\pm 2.9 \text{ ms}$ in seven days. A detailed specification of the OCXO MQ oscillator is available in Table 3.4.

Table 3.7: Different antenna types which are compatible to the Meinberg receiver [10]

Receiver Type	Signal Type	Value	Connector
Meinberg GPS IF, 12-channel	IF (Meinberg Antenna)	15 V	DC BNC
Meinberg GNS-UC GPS/Galileo IF	IF (Meinberg Antenna)	15 V DC	BNC
GNSS (GPS, GLONASS, Galileo, BeiDou), 72-channel	L1/E1/B1 band	5 V DC	SMA

Since the GM is located within the building, the coax cable length between receiver (GM) and antenna could reach up to 100 m. Meinberg provides an Equation 3.1 to calculate the maximum possible coax cable length depending on the cable type. The IF α_1 , is the highest frequency transmitted on the cable [39]. According to the Meinberg homepage the maximum attenuation of 30.4 dB at given IF of 35.42 MHz between antenna and receiver must not be exceeded. The RG-58 coax cable that Meinberg offers have a attenuation of $\alpha_1 = 17 \text{ dB}/100 \text{ m}$ at 100 MHz. Therefore, according to Equation 3.1 the maximum cable length between antenna and receiver is 300 m.

$$\alpha_2 = \alpha_1 \sqrt{\frac{f_2}{f_1}} \quad (3.1)$$

$$\alpha_2 = 17 \text{ dB}/100 \text{ m} \sqrt{\frac{35.42 \text{ MHz}}{100 \text{ MHz}}} = 10.2 \text{ dB}/100 \text{ m} \quad (3.2)$$

An RG-213 coax cable with an attenuation factor of 7 dB/100 m would achieve a maximum length of 700 m but is also less flexible to handle within small pipes. Thus, as a risk-

mitigation measure and for better cable flexibility a 100 m RG-58 coax cable is used for the PTP Backbone.

3.2.2 PTP Switch

The PTP Backbone should be capable of connecting multiple devices that could benefit from a global clock synchronization. Therefore the two PTP Ethernet output ports from the microSync RX102 are not enough to fulfill this requirement. A dedicated switch that supports the mentioned PTPs is necessary to distribute the GMC to every device and act as a transparent or boundary clock within the PTP network.

Besides the mandatory and optional requirements from Table 3.5 the PTP switch should also provide PoE to avoid the urgency of an external power supply for the SyncBoxes. CISCO is one of the biggest vendors for high-performance switches and offers a wide selection of switches that support IEEE 1588. Due to mentioned timing constraints and long delivery times of over 110 days, purchasing re-manufactured CISCO components from certified partners offered a great opportunity regarding short delivery times, and also high-quality products with a warranty.

The re-manufactured IE40004GC4GP4GE switch from CISCO, fulfills all optional and mandatory requirements and was delivered within a few weeks. The matching 170 W power supply module PWR-IE170W-PCAC was also available at the same certified CISCO partner. The IE4000 has eight combo ports, SFP (fiber) or RJ45 (10/100/1000 Mbps) and four additional PoE GigaByte-Ethernet 125 W ports. It supports PTPs like IEEE 1588, 802.1AS, and Power Profile (C37.238-2011/2017 and IEC 61850-9-3). But the only PTP profile that supports network protocol 802.3 (layer 2) is the Power Profile.

3.2.3 PTP Hardware Configuration

To establish the PTP network every component need to be configured with the same PTP profile, network transport protocol, PTP domain, and delay mechanism. Those configuration parameters can be adjusted via telnet, web browser, or the GUI Meinberg Device Manager, with the related IP address from Table 3.8. In this section the most important configurations will be explained. The GTSYNC card only accepts PTP synchronization messages using network protocol 802.3 (layer 2). Therefore, all other components need to use that network protocol. Since the CISCO IE4000 switch only provide 802.3 network protocol in the PTP Power Profile the only common PTP profile is IEC 61850-9-3.

Grandmaster microSync RX102

The Meinberg Device Manager is an application to configure and monitor Meinberg devices like the microSync RX102 or the SyncBox. During this work the Meinberg Device Manager firmware version Southbotton 2020.11.4-u 5cfab0a5 and microSync 2.71 are used. After adding a physical device via IP/Media Access Control (MAC) address or

Table 3.8: MAC and IP addresses of all components within the PTP network

Device Name	IP Address	MAC Address
Cisco IE4000	192.168.2.11	70:61:7b:a2:d7:01
microSync RX102	192.168.2.2	ec:46:70:0c:ee:ea
OptiPlex 9010	192.168.2.20	b4:96:91:62:34:32
PowerEdge R640	192.168.2.25	e4:3d:1a:4d:50:6c
RTDS GTSYNC	192.168.2.14	00:50:c2:4f:96:9f
SyncBox1	192.168.2.31	ec:46:70:0b:5f:53
SyncBox2	192.168.2.32	ec:46:70:0b:5f:52
Wireshark	192.168.6.20	N/A

auto search, every added device is accessible via the drop-down menu at the top. The GUI is structured in two main sections, a *Config* section on the left and a *Status* section on the right side of the screen. Under Config/References the GNSS satellite systems is selected as main, and NTP as a second time source. By selecting the IEC 61850-9-3 profile in the Config/PTP (IEEE1588) section most of the parameters like P2P delay mechanism, IEEE 802.3 network protocol, and International Atomic Time (TAI) time format are pre-selected and not changeable. Therefore it is important to verify that devices in the PTP Backbone are capable of detecting the TAI time format and not only expecting UTC. PTP domain is set to 0 and should match with every other device in the PTP Backbone.

Monitoring: Meinberg Device Manager offers different monitoring tools in the *Status* section. Reference clock status should be, *clock synchronized and Oscillator warmed up*, the estimated time quality of 100 ns is also displayed in this section. Reference Antenna status should be, *accepted as Master*, and *locked with low Jitter* flag. Table 3.9 shows a certain moment of locked and in view satellites. The position, index number (SVNO) of the satellites, and the Carrier to noise density ratio (C/NO) of each satellite are also visible. The *Event Log* is tracking every change of Antenna or synchronization state and categorizes them into three levels (Info, Warning, and Critical). The PTP (IEEE1588) Status sections offer an overview of PTP parameters like current UTC offset, total transmitted and received packets.

SyncBox

Analog to the microSync configuration, the SyncBox can be configured via Meinberg Device Manager. In the current Device Manager version 6.4 the SyncBox has no pre-defined IEC 61850-9-3 profile. Instead, the *Custom* profile can be selected, in combination with Multicast Slave role, IEEE 802.3 Network Protocol, PTP Domain Number 0 and P2P Delay Mechanism (see Table 3.10). As a reference source PTP should be selected instead of NTP (default). For a trustworthy synchronization the Clock, Reference and PTP Status can be tracked as explained in the microSync RX102 configuration section.

Table 3.9: List of available GPS/Galileo satellites.

GNSS	SVNO	Elevation [°]	Azimuth [°]	C/NO [dBHz]	Status
GPS	3	46	266	20	Locked
GPS	4	37	320	29	Locked
GPS	9	5	192	0	In View
GPS	16	73	192	18	Locked
GPS	22	36	114	34	Locked
GPS	26	67	46	35	Locked
GPS	27	16	162	22	Locked
GPS	29	7	45	22	Locked
GPS	31	33	55	38	Locked
GPS	32	16	125	33	Locked
Galileo	1	72	233	10	Locked
Galileo	4	43	320	25	Locked
Galileo	19	73	112	19	Locked
Galileo	21	54	38	33	Locked
Galileo	27	4	42	7	In View
Galileo	31	17	224	12	Locked

Table 3.10: Meinberg SyncBox PTP configuration.

PTP Settings		Ref. Source Settings	
Role:	Multicast Slave	Ref. Source 1:	PTP
Profile:	Custom		
Network Protocol:	IEEE 802.3		
Domain Number:	0		
Delay Mechanism:	P2P		

PTP Switch

The access of the CISCO switch via telnet offers full configurations possibility. The GUI via the web browser on the other hand is more convenient for monitoring during run time. Listing 3.1 provides the necessary commands for complete configuration. According to the CISCO PTP Software Configuration Guide [40] IEC 61850-9-3 is available in the *power profile* (line 5). When the switch is in power profile mode, only P2P-transparent clock (line 6) is supported. In addition, *allow-without-tlv* command (line 7) is necessary to guarantee proper operation. To track network traffic through the switch, Switched Port Analyzer (SPAN) can be configured to pass through all ports (line 9) to one single monitoring port (line 11). The flag *encapsulation replicate* is mandatory to monitor Layer 2 traffic. Wireshark can be used to track the whole PTP network traffic via Port 8 if the CISCO switch.

```

$telnet 192.168.2.11 1
//Enter Login Credentials 2
IE4000>en 3
IE4000#configure terminal 4
IE4000(config)ptp profile power 5

```



```

IEC4000(config)ptp mode p2ptransparent      6
IEC4000(config)ptp allow-without-tlv      7
IEC4000(config)monitor session 4 source interface 8
    Gi1/1 - 3 , Gi1/5 , Gi1/7 , Gi1/9 - 10  9
IEC4000(config)monitor session 4 destination 10
    interface Gi1/8 encapsulation replicate 11
IE4000#show running-config                12
IE4000#copy running-config startup-config 13

```

Listing 3.1: PTP switch CISCO IE4000 configuration.

GTSYNC Card

The GTSYNC card can be configured directly via NovaCor, or the GTWIF card (in case of a GPC card instead of NovaCor). Listing 3.2 shows the major parameters configurable via telnet (line 1). Simple Network Protocol (SNTP) is another time synchronization protocol, to avoid ambiguity SNTP is deactivated by setting this parameter to 0.0.0.0 (line 9). GTSYNC card can only operate in Slave mode (line 10) when the master provides at least 1 μ s time accuracy, which Meinberg microSync RX102 fulfills. PTP Domain should be set to the same value as GMC to establish a connection (line 12). Advertised time source (line 14) is a status flag and indicates that the time source is actually the GPS antenna and not any other clock source. The GTSYNC card assumes UTC as time format via PTP. Since microSync RX102 IEC 61850-9-3 profile use exclusive TAI format, further data processing explained in Section 4.3.1 is necessary.

```

$telnet rack12.caps.fsu.edu                1
> gtwif                                    2
PER > 64                                    3
Current GTSYNC configuration:              4
Ethernet port          : Copper RJ45      5
IP address             : 192.168.2.14     6
Subnet mask            : 255.255.255.0    7
Gateway IP address    : 192.168.2.1      8
SNTP server IP address : 0.0.0.0         9
PTP mode               : Slave-only      10
PTP profile            : IEC/IEEE 61850-9-3 Power Utility Profile 11
PTP domain             : 0               12
Sync mode              : IEEE 1588       13
Advertised time source : \gls{gps}      14

```

Listing 3.2: PTP configuration of the GTSYNC card

NIC Intel i350

Two different DELL computers are involved in this setup for performance validation which will be further explained in Chapter 4. The Dell PowerEdge R640 with an Intel Xeon Gold 6210U CPU (2.5 GHz) and 96 GB Ram has Linux Red Hat Enterprise 7.9 installed. On the other hand, the Dell OptiPlex 9010 with an Intel Core i7-3770 CPU (3.4 GHz) and 8 GB Ram has Linux Ubuntu 20.04 LTS installed. Both machines have the

Intel i350 NIC installed. Linuxptp offers two specific commands to synchronize the NIC clock to the GMC, `ptp4l` and `phc2sys`. The `ptp4l` command is responsible for the PTP to synchronize the specified NIC port. Every other clock in the OS or NIC port that needs to be synchronized has to be configured via the `phc2sys` command. For example, `CLOCK_REALTIME` is a reference to most user applications and therefore needs to be synchronized separately. Listing 3.3 shows an example configuration according to the flags explained in Table 3.11. The `-w` (wait) flag is important to guarantee that all configured clocks/ports are only synchronized when the `ptp4l` command is running and in calibrated state.

```
sudo ptp4l -i p2p1 -P -2 -s -m
sudo phc2sys -c CLOCK_REALTIME -s p2p1 -w -m
sudo phc2sys -c p2p2 -s p2p1 -w -m
```

Listing 3.3: NIC configuration with `ptp4l` and `phc2sys` commands

Table 3.11: Flag description of `ptp4l` and `phy2sys` command.

<code>ptp4l</code>	Flag	Description	<code>phc2sys</code>	Flag	Description
	<code>-P</code>	P2P		<code>-s [dev]</code>	master clock
	<code>-2</code>	IEEE 802.3		<code>-c [dev]</code>	slave clock
	<code>-H</code>	hardware ts		<code>-w</code>	wait for <code>ptp4l</code>
	<code>-s</code>	slave only mode		<code>-m</code>	print msgs to stdout
	<code>-m</code>	print msgs to stdout			
	<code>-i [dev]</code>	interface device			

Table 3.12 shows an example output of a `ptp4l/phc2ys` command. In this case, NIC port `p2p1` is connected to the PTP network, and `CLOCK_REALTIME` (`clock_rt`) is synchronized to the PTP reference. After an initialization phase, the TAI time format is updated to UTC according to the current offset. The first column represents the timestamp of the respective output in seconds since Epoch. Offset represents the time in nanoseconds between master and slave. In the case of `ptp4l`, between GMC and NIC port `p2p1`. Both services have three different states: unlocked (`s0`), clock step (`s1`), and locked (`s2`). After the system is in locked state, `phc2sys` service is switching from *Waiting for ptp4l...* to operating mode. Freq indicates the frequency adjustment of the respective clock in Part per Billion (PPB) to synchronize the slave with the master clock. Delay is an estimated synchronization message delay between master and slave in nanoseconds. Path delay of `ptp4l` depends on cable length between devices within the PTP network. Delay from `phc2sys` can vary by a factor of 10 depending on the hardware architecture of the NIC. In general `ptp4l` and `phc2sys` indicate a synchronization to the GMC. However, this quality degrades for applications as their software stacks cause additional jitter. Thus, it is one part of Chapter 4 to validate synchronization during

a specific user application. The explained output from the `ptp4l` command is also an indicator for successful communication between the NIC and the PTP Backbone.

Table 3.12: Example output from `ptp4l` and `phc2sys` command.

```

ptp4l[2561913.014]: selected /dev/ptp2 as PTP clock
ptp4l[2561913.034]: port 1 (p2p1): initializing to listening on init_complete
ptp4l[2561913.034]: port 0 (/var/run/ptp4l): initializing to listening on init_complete
ptp4l[2561913.034]: port 0 (/var/run/ptp4lro): initializing to listening on init_complete
ptp4l[2561913.308]: port 1 (p2p1): new foreign master ec4670.ffe.0ceeea-1
ptp4l[2561914.308]: updating UTC offset to 37
ptp4l[2561915.309]: selected best master clock ec4670.ffe.0ceeea
ptp4l[2561915.309]: port 1 (p2p1): listening to uncalibrated on rs_slave
ptp4l[2561916.048]: master offset      -32031  s0  freq   36029  path delay  296
ptp4l[2561917.049]: master offset      -32050  s1  freq   36010  path delay  294
ptp4l[2561918.047]: master offset           -3   s2  freq   36007  path delay  293
ptp4l[2561918.048]: port 1 (p2p1): uncalibrated to slave on master_clock_selected
ptp4l[2561919.049]: master offset           7   s2  freq   36013  path delay  294
ptp4l[2561920.046]: master offset          15   s2  freq   36030  path delay  293
ptp4l[2561921.047]: master offset          -14  s2  freq   36023  path delay  292

phc2sys[2561917.049]: Waiting for ptp4l...
phc2sys[2561918.046]: clock_rt phc offset  -53252  s2  freq   -3957  delay     841
phc2sys[2561919.047]: clock_rt phc offset   -453   s2  freq   -4562  delay     834
phc2sys[2561920.048]: clock_rt phc offset    96   s2  freq   -5015  delay     834

```

PTP Monitoring

Multiple status LEDs and status messages in Meinberg Device Manager or Wireshark are available to verify a successful connection of a device after connecting it to the PTP Backbone and configuring it properly. The status LEDs of the GM microSync RX102 can be interpreted according to Figure 3.13. In addition, in the Meinberg Device Manager section *Status/PTP (IEEE1588)* the number of transmitted and received PTP packages are visible. This signals, that the GM and at least one additional device are correctly configured and a PTP communication is established.

The Meinberg SyncBox can also be monitored via status LEDs (Figure 3.14) or Meinberg Device Manager. In section *Status/PTP* several parameters about the current PTP connection are available. For example, the calculated path delay between GM and SyncBox and the corresponding time *Offset* in nanoseconds. Also, the *Time Source* flag indicates if GM is using the GPS system as time source instead of its internal oscillator. In addition, since TAI is the selected time format, the *UTC Offset* of 37s is a sign for a correct time format conversion. It is important to mention that, all status LEDs can be green although the internal Oscillators are not warmed up. It can take several hours for the warm-up process, depending on environment's temperature. Since the clock accuracy is also temperate dependent, it is necessary to verify that warm-up phase is completed.

Table 3.13: GM microSync RX102 description of status LEDs [10].

CPU		REC:	
R (Receiver)		Fail	
green:	The reference clock (e.g. build-in GNS181-UC) provides a valid time	red:	No synchronization
red:	the reference clock does not provide a valid time		
T (Time Service)		Ant	
green:	NTP is synchronized to the reference clock, e.g. GNS181-UC	green:	Antenna connected and clock is synchronized
red:	NTP is not synchronized or switched to the "local clock"	red:	No synchronization resp. no antenna connected or short circuit on the antenna line
N (Network)		Init	
green:	All monitored network interfaces are connected ("Link up")	green:	"warmed up" - oscillator is adjusted
red:	At least one of the monitored network interfaces is faulty	blue:	Initialisation phase
A (Alarm)			
off:	No error		
red:	General error		

Listing 3.2 already explained a quick method of proving the PTP connection by checking the *Advertised time source (GPS)*. RTDS provides the software tool RSCAD as GUI for real-time simulations. To make timestamps from the GTSYNC card available during a simulation, the GT_SYNC module has to be added. After compiling a RSCAD simulation including a GT_SYNC module, the CRTSECD and CRTNSEC signals are available. These two signals include the current Epoch timestamps from the GTSYNC card according to the GMC. A detailed description about signal and flag configuration will be explained in Chapter 4.3.1 Figure 4.11.

A successful connection between the PTP Backbone and the Intel i350 NIC can be directly verified via *ptp4l* and *phc2sys* command. Table 3.12 is an example of a successful synchronization of the NIC PTP slave port (*ptp4l*), and an additional clock (*phc2sys*). Two parts are relevant here in terms of accuracy. First the *phc2sys* service tries to synchronize the *CLOCK_REALTIME* to the synchronized slave port. But at the beginning (timestamp 2561917.049) the *ptp4l* service is not in a locked *s2* state. So this behavior guarantees that as long as the *ptp4l* slave port is not in locked state, the *phc2sys* state stays in *Waiting for ptp4l*. The other interesting aspect is, that both PTP services have a short warm-up phase before the offset is stagnating at a certain value,

Table 3.14: Description of status LEDs regarding Meinberg SyncBox.

LED Indicators		
LI - Link:	lights up in the same color as SP-Speed, on or off, if no link is available	
SP - Speed:		
	red	no link available
	yellow	10Mbit
	green	100Mbit
IN - Input:		
	red	no reference
	yellow	reference is available
	green-blinking	synchronous
	green	oscillator has locked
ST - Status:		
	blue	during initialization
	green	normal operation
	red	error

usually a few nanoseconds.

3.2.4 Final Hardware Setup

Figure 3.3 demonstrates the final setup of the PTP Backbone. All components are designed to fit into a rack tower. The Dell PowerEdge R640 server and the OptiPlex Tower at the bottom of the Rack-Tower. The two Syncboxes are also mounted into the rack during this test scenarios and will be placed next to scopes and the DUT during real PHIL/CHIL experiments. The Meinberg microSync RX102 1 U Rack-size, the CISCO IE4000 switch in combination with related power supply on the top. The white device on the top, is an Ethernet-CAN-Gateway from Peak-System. Since some DUTs might support CAN bus, this device was also included in the design concept developing process. Nevertheless, this device is currently not included in any experiment but has the potential for benefits in future projects. All RTDS interface cards like the mentioned GTSYNC, GTNET, or GTWIF cards are centralized together in the RTDS Racks.

The GNSS Antenna has a complete rooftop mounting set including a surge voltage protection in case of lightning and a 100 m cable to mount it outside. Due to convenience reasons and higher accessibility during experiments presented in this Thesis, the GNSS Antenna is mounted temporarily inside but still connected with the 100 m coax cable between Antenna and microSync receiver. As shown in Table 3.7 this had no negative influence on the experiment. The Event-Log from the Meinberg microSync tracked not a single connection loss during the whole testing period. Nevertheless, it can be assumed

3. SYSTEM ARCHITECTURE / IMPLEMENTATION

that more satellites would be insight with an appropriate roof top mounting, but also increased the chance of damage caused by animals or weather. A list of all components used for experiments or for future projects are listed in Table 3.15.



Figure 3.3: Core components of the PTP Backbone.

Table 3.15: All components used or available.

Product	Company	Article Number	Description
Receiver	Meinberg	microSync RX102	GNS-UC Receiver, 10 Series, OCXO MQ Oscillator
Power Supply	Meinberg	microSync AD10	100-240 VAC / 100-200 VAC
SFP Module	Meinberg	ABCU-5740RZ	1000Base-T, Copper Gigabit Ethernet
SFP Module	Meinberg	AFBR-5710PZ	1000 BaseSX, Multimode 850 nm, Gigabit Ethernet
Surge-Suppressor	Meinberg	MBG S-PRO	MBG S-PRO Surge Suppressor
SyncBox	Meinberg	SyncBox/N2X/PP-3	Input: PTP, Output: IRIG, 10 MHz, PPS, DCF77
Coax Cable	Meinberg		RG58 5 m, BNC (male/male), used for scopes
	Meinberg		RG58 5 m, N-Norm(male/male), between antenna and Surge Suppressor
	Meinberg		RG58 100 m, N-Norm(male) / BNC(male), between Surge Suppressor and GM
Switch	CISCO	IE40004GC4GP4GE	PTP, 12 Ports - Manageable - Gigabit Ethernet 10/100/1000 Base-TX
Power Supply	CDW	PWR-IE170W-PCAC-RF	170 W PoE AC-DC
Server	DELL	PowerEdge R640	Intel Xeon Gold 6210U 2.5 GHz, 92 Gb RAM
Tower	DELL	OptiPlex 9010	Intel Core i7-3770 CPU (3.4 GHz), 8 GB Ram
SFP Module	FS.com	SFP-GB-GE-T	10/100/1000 BASE-T, RJ-45
SFP Module	FS.com	SFP1G-SX-85	1000 Base-SX, 850 nm
SFP Module	FS.com	SFP-10G-T	10 GBase-T, RJ-45
SFP Module	FS.com	SFP-10GSR-85	10 GBase-SX, 850 nm
Rack-Mounting	Amazon		Screws, DIN Rail Kit, 1U 19-inch 4 post rack
Available but no present			
FPGA Board	Mouser	424-Netfpga-Sume	
Ethernet/ CAN Gateway	PEAK	IPEH-004012	



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

System Performance Validation

Chapter 3 introduced the developed PTP Backbone to synchronize clocks of HIL testbed components using Linuxptp. In general a user application does not have direct access to precise timestamps of the Linuxptp service. To demonstrate how user applications can take advantage of a clock-synchronized network, the following use cases can be defined:

- RTDS Validation Process:
Demonstrating the possibility to use synchronized time reference within a HIL simulation in RSCAD. In particular, using this time reference for one-way path delay measurements between RTDS and a Linux machine synchronized to the PTP Backbone.
- CORE Validation Process:
Using hardware timestamps of a synchronized NIC to measure latency of an emulated switch in CORE compared to a real hardware switch. Results can be compared to related work from Ogilvie et al. [6] that uses ping command for the latency measurement. Thus, this use case represents the benefits between synchronized and not synchronized systems.
- SyncBox Validation Process:
Validation of clock stability between two SyncBoxes, to demonstrate the possibility to synchronize devices which do not support PTP.

The lower part of Figure 4.1 visualizes the PTP Backbone introduced in Chapter 3, and the upper part the measurement data network. RTDS and the two Linux computers introduced in Chapter 3.2.3 are sharing the same time base, due to the PTP Backbone. By sending UDP messages with a unique payload, the moment of sending and receiving packages from one-way messages can be tracked precisely and stored in a common file system. During the post-processing, collected data is analyzed to calculate path delays,

and clock drifts. The data network can be categorized into two parts, CORE and RTDS validation, and the developed UDP script as the common measuring method. These two parts of the validation procedure and the developed UDP measurement script will be explained in detail in the following sections.

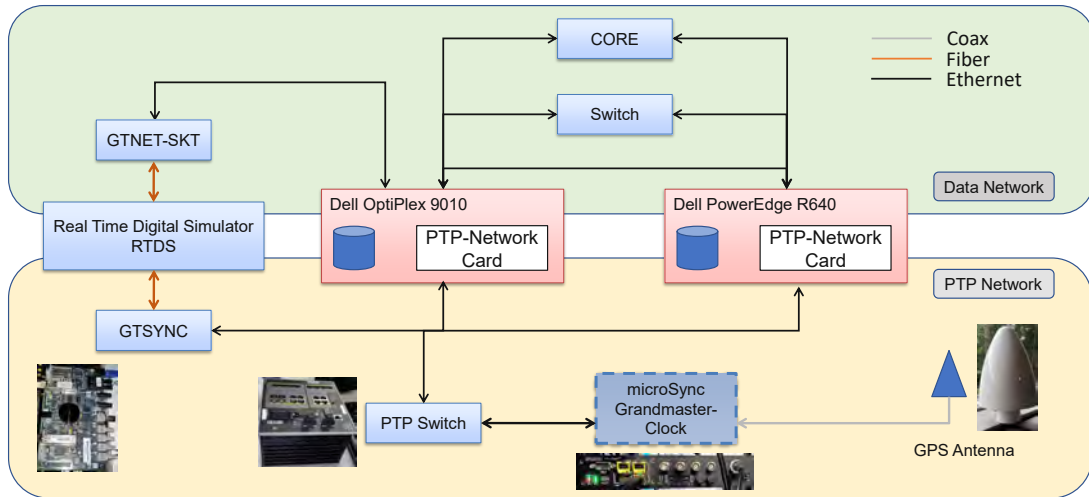


Figure 4.1: Measurement setup to validate system performance.

4.1 UDP measurement script

One possible use case to take advantage of synchronized clocks within a network is the possibility of one-way path delay measurements. Therefore, a precise latency measurement method was developed based on UDP messages that support hardware time stamping. This further increase the time stamping precision compared to software-based solutions.

The open-source package `Linuxptp` is using UDP communication in combination with hardware time stamping to achieve accuracy in nanosecond resolution. This led to the basic communication protocol in terms of path delay measurement during the following validation processes. Different example solutions and detailed documentation about UDP message time stamping are available in the literature [41]. Thus, the programming language C was selected for measuring hardware timestamps, and Python for data post-processing.

The basic idea is to send a UDP message from the Dell OptiPlex 9010 machine to the Dell PowerEdge R640 machine and capture the hardware timestamp from the moment of transmitting (TX). Analog the Dell PowerEdge R640 computer is capturing the timestamp of the receiving message (RX). Both machines can save those data to the common shared file server. Since both NICs are synchronized to the GMC via the PTP Backbone the path delay is the difference between RX and TX timestamp. Another scenario is sending UDP from Dell Optiplex 9010 messages to RTDS and receive messages on the same

machine. This measurement setup will be explained in detail in Chapter 4.3.1, but the fundamental methods are explained in this Chapter.

Even when the sending and receiving routine is executed on different computers, it was more convenient to implement code in one common file named *rtds.c*. By executing the c-file with an additional argument like -s (sending), -r (receiving), or -sr (sending and receiving on the same machine). Different sequences of the code are executed, illustrated in Figure 4.2. Each part of the code is structured similarly. First, configuring and initializing the socket, starting the sending/receiving routine, and storing the data into a text file for post-processing. These three parts of the code will be explained in detail in the following sections. The sending and receiving part on the same machine is an individual measurement required for the RTDS validation procedure which will be introduced in Chapter 4.3.1.

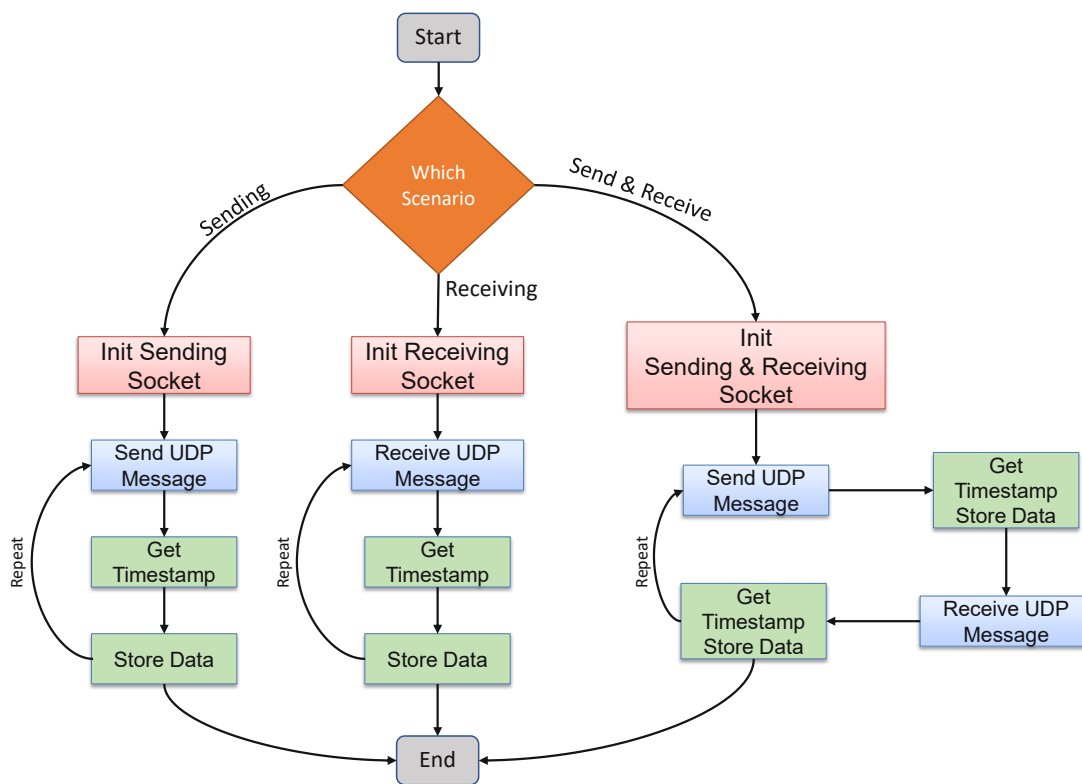


Figure 4.2: Flowchart of the UDP measurement script

Before a UDP connection between two devices can be established, and UDP sockets can hardware timestamp in- and outgoing packets, two requirements have to be met. First, by installing the Linuxptp package the terminal command *hwstamp_ctl* is available. The following command enables hardware timestamps for a specific interface.

```
hwstamp_ctl -i [interface] -r 1 -t 1
```

This needs to be done for every interface that runs a UDP measurement script that requires hardware timestamps. Second, the firewall of each Linux machine needs to be configured to allow communication on the selected port.

4.1.1 UDP Socket Configuration

Several steps are necessary to configure a socket to send and receive UDP messages and enable time stamping functionalities (Listing 4.1). The `socket()` method (Line 1) creates a communication endpoint according to the given arguments, *domain*, *type*, and *protocol*. The `AF_INET` address family argument enables IPv4 Internet protocols and the `SOCK_DGRAM` UDP messaging. *Protocol*, the third argument defines the communication protocol used depending on the selected *domain* and *type*. In this particular case, only one protocol exists so this argument could be set to 0, or manually set to `IPPROTO_UDP` [42].

The `setsockopt()` method (Line 4) allows to set specific socket options to enable software/hardware timestamps of packets. The *skt* argument contains the file descriptor of the created socket. `SOL_SOCKET` defines the protocol *level* where the option specified from `SO_TIMESTAMPING` (*option_name*) is located. The array *ts_opt* (Line 3), in this case, contains the required flags to enable hardware time stamping. Analog, software time stamping can be enabled by replacing `*_HARDWARE` with `*_SOFTWARE` and `*_RAW_HARDWARE` with `*_SOFTWARE` [43]. Software timestamps are available in most NIC, and therefore also used in the `ping` command [44].

After a socket is created and required options are set, the method `bind()` is required to bind the socket to a local interface with the corresponding IP address and port number. The structure *recv_addr* of type `sockaddr_in` contains all required information like socket domain, IP address, and port. The method `inet_aton()` converts IP addresses from numbers-and-dots to binary form, and `htons()` from unsigned integer to network byte order (Lines 7 and 8).

```

skt = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);           1
ts_opt = SOF_TIMESTAMPING_RX_HARDWARE |                 2
        SOF_TIMESTAMPING_TX_HARDWARE | SOF_TIMESTAMPING_RAW_HARDWARE; 3
setsockopt(skt, SOL_SOCKET, SO_TIMESTAMPING, &ts_opt, sizeof(ts_opt)); 4
                                                                    5
recv_addr.sin_family = AF_INET;                          6
inet_aton(ip_addr, &recv_addr.sin_addr);                 7
recv_addr.sin_port = htons(PORT);                       8
bind(skt, (const struct sockaddr *)&recv_addr, sizeof(recv_addr)); 9

```

Listing 4.1: Fundamental methods to initialize a UDP socket.

4.1.2 Sending and receiving UDP packets

To access timestamps later on, control messages (also called ancillary data) that can store information like timestamps, and related interface are necessary. The methods `sendmsg()` and `recvmsg()` (for sending and receiving) can send/receive such control

messages to/from the socket. They require three arguments, *sockfd*, *msghdr*, and *flags*. The *sockfd* flag holds the file descriptor of the associated socket. The structure *msghdr* (Listing 4.2) contains among others the payload (**msg_iov*), the destination IP address (**msg_name*), and the control message (**msg_control*). If the message size exceeds a certain length the message will not be transmitted. The **sendmsg()** method in this case return the error EMSGSIZE. In terms of sending a packet, the third argument *flag* is set to 0 since no further configuration is necessary [45].

```
struct msghdr {
    void            *msg_name;           /* Optional address */
    socklen_t       msg_namelen;        /* Size of address */
    struct iovec    *msg_iov;           /* Scatter/gather array */
    size_t          msg_iovlen;        /* # elements in msg_iov */
    void            *msg_control;       /* Ancillary data, see below */
    size_t          msg_controllen;     /* Ancillary data buffer len */
    int             msg_flags;          /* Flags (unused) */
};
```

Listing 4.2: Definition of the msghdr structure [45]

If a socket receives a message and time stamping options are set properly, ancillary data including timestamps are created. Therefore, the **recvmsg()** can be used to retrieve this information from the socket.

By calling the **sendmsg()** method, the packet will be hand over into a sending queue. Thus, the timestamp of actual sending the message is not available at this point and the control message remains empty. Instead, outgoing packets are also looped back into the socket's error queue with the send timestamps attached. By executing the **recvmsg()** with the *MSG_ERRQUEUE* flag, this method return relevant metadata of the originally transmitted message including timestamps [41]. Listing 4.3 shows an example of a sending and receiving socket with ancillary data containing timestamps. The unique string *UDP_n* in the payload, where *n* is an integer number starting from 1, allows data correlation during the post-processing. With those RX and TX timestamps, path delay between both devices running the script can be calculated.

The method **clock_gettime()** saves the current time from the related local clock to the given file descriptor. *CLOCK_REALTIME* is a clock that can be modified or adjusted by certain services like NTP or Linuxptp. Its time represents the current seconds and nanoseconds since Epoch. *CLOCK_MONOTONIC_RAW* on the other hand, can not be modified or tuned in terms of frequency. So these two timestamps right before sending a message and after receiving one can be used as a reference compared to path delay calculated from hardware timestamps.

```
\\Sending Socket
clock_gettime(CLOCK_REALTIME, &pre_ts_rt);
clock_gettime(CLOCK_MONOTONIC_RAW, &pre_ts_mraw);
sendmsg(skt, &msg, 0);
while(1){
    recvmsg(skt, &msg, MSG_ERRQUEUE);
```

```

    break;
}

\\ Receiving Socket
recvmsg(skt, &msg, 0);
clock_gettime(CLOCK_MONOTONIC_RAW, &post_ts_mraw);
clock_gettime(CLOCK_REALTIME, &post_ts_rt);

```

Listing 4.3: Fundamental commands to send and timestamp a transmitted message.

4.1.3 Accessing timestamps via a control messages

Control messages (also called ancillary data) contain additional information (like timestamps) about the interface (socket) that received incoming packets and are not part of the socket payload. As mentioned above, ancillary data can be received by calling `recvmsg()`. This structure contains various information in different socket *levels*. Therefore, macros from the `cmshdr` struct can be used to iterate through the data [46], Listing 4.4 illustrates this procedure. Line 3 is searching for ancillary data that matches the *level* and *option_name* with configured socket options. When this condition is met, `CMMSG_DATA(msg)` contains the corresponding timestamps and copies them to `struct scm_timestamping *ts`. Depending on the socket option `ts[0]` holds software timestamps and `ts[2]` hardware timestamps. In general hardware timestamping is the preferred configuration due to higher precision.

```

struct cmsghdr *cmsg;
for (cmsg = CMSG_FIRSTHDR(msg); cmsg; cmsg = CMSG_NXTHDR(msg, cmsg)) {
    if (cmsg->cmsg_level == SOL_SOCKET && cmsg->cmsg_type == SO_TIMESTAMPING){
        struct scm_timestamping *ts = (struct scm_timestamping*)CMSG_DATA(cmsg);
    }
}

```

Listing 4.4: Reading timestamps from ancillary data.

Every single sent and received UDP message has corresponding hardware timestamps, timestamps from `CLOCK_REALTIME` and `CLOCK_MONOTONIC_RAW`, and the unique message number. To avoid loss of data during run-time, and reduce used workspace memory, each dataset is saved into a text file after each sending/receiving sequence.

4.1.4 Linuxptp data

The Linuxptp project is an open-source project and thus still might have some bugs and issues. As explained in Chapter 2.1 the PTP requires four messages to calculate the path delay and clock offset between two devices. Due to timing constraints and packet loss, the `ptp4l` service might switch to an uncalibrated state. In this case, the clocks are not synchronized to the GMC anymore and could drift apart. Usually this uncalibrated state last between 20 and 60s. During this period, hardware timestamps from the introduced UDP measurement script are not that accurate. However, we can track the non-calibrated phases via the output of the `ptp4l` and `phc2sys` programs. To determine which datasets are *valid* (captured during calibrated state), the Linuxptp

package was minimally modified. Every time the output from the `ptp4l` and `phc2sys` service (see Table 3.12) is printed, the same line is also saved to a text file. Every line has an additional column including timestamps from local `CLOCK_MONOTONIC_RAW`. Therefore, the data frame from the developed UDP script can be merged with the data frame from the `ptp4l/phc2sys` data frame. This allows the categorization of *valid* and *invalid* datasets and adds additional information to every UDP message.

4.2 CORE Validation Process

In Chapter 2.3.3 the work from Ogilvie et al. [6] is summarized. Ogilvie et al. demonstrated the behavior of emulated networks in CORE compared to real networks. Therefore, they measured latency of a real hardware switch using ping command and compared it to a virtual switch within CORE. This procedure relies on software timestamps and end-to-end RTT measurements. Due to synchronized clocks, the developed PTP Backbone offers the opportunity to use one-way path delay measurements with nanosecond resolution based on hardware timestamps. Therefore the demonstrated setup from Ogilvie et al. [6] acts as a reference to compare the results of these two measurement methods.

4.2.1 CORE Measurement Setup

To validate the performance of CORE, three different latency measurement setups between the two Linux machines are developed (see right part of Figure 4.3). To use synchronized clocks in a user application, the specific ports used for the measurement has to be synchronized to the PTP Backbone. Therefore, Port 1 of each Intel i350 NIC from both Dell computers are synchronized to the GMC via `ptp4l` command from the `Linuxptp` package. Every Port required for the measurement is synchronized to Port 1 via `phc2sys` command. Table 3.12 shows the required command arguments and related output.

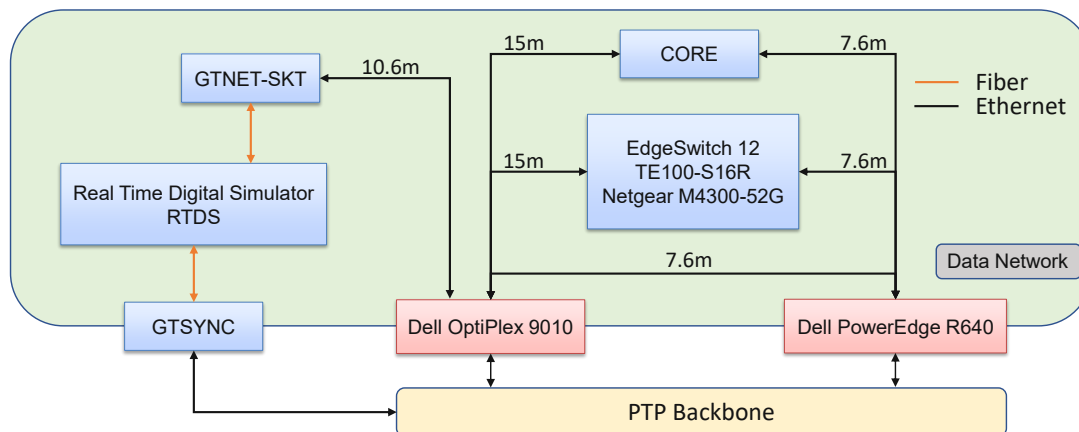


Figure 4.3: Measurement setup during the validation process.

In phase one (direct connection), additional latency and other dependencies can be reduced to a minimum. Thus, it is possible to calculate the theoretical propagation delay of a specified package within the Ethernet cable with a defined length. This value can be compared with the measured latency from the hardware timestamp UDP script and verify the accuracy and validity of the developed UDP measurement script.

In the second phase, the latency of three different hardware switches are measured with the now approved UDP script. The Netgear M4300-52G switch is the same model as the one used in the reference measurement [6]. This allows a accurate comparison between both concepts. For further reference data, the latency of a 1000 Mbps (EdgeSwitch 12) and a 100 Mbps (TE100-S16R) switch are also investigated. The obtained data of these three hardware switches serve as a reference for the emulated switch in CORE.

The third phase includes latency measurements of a emulated switch within core. The possibilities of configuration in CORE is limited. In the current version 6.5.0 the `Ethernet switch` model has no further configurable parameters. The `links` between modules have parameters like, Bandwidth, Delay, Jitter, Loss and Duplicate. As an initial test, it is relevant to figure out the minimal achievable latency of an emulated switch, see Figure 4.4. By setting the delay to zero the minimal latency for 1000 Mbps connection can be emulated. From this minimal reference, it is possible to increase the delay and compare it with the measured latency from the hardware timestamped UDP script. Chapter 4.2.2 will present all results and comparisons between real and emulated switches.

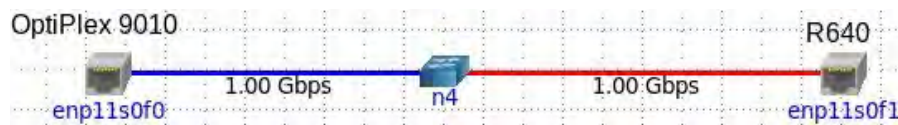


Figure 4.4: Emulated switch within the CORE simulation.

4.2.2 CORE Validation Results

As explained in Section 4.1.4 due to timing requirements, the Linuxptp service might switch into an uncalibrated state for several seconds. Since this behavior might be part of every real scenario, it is important to visualize ideal versus real circumstances. For the following section data during an uncalibrated state are categorized as *invalid*, and synchronized data as *valid*.

Direct Connection

In this section, the developed UDP measurement script is used to measure one-way path delay of a direct Ethernet connection between Optiplex 9010 and PowerEdge R640. The packet size of each transmitted UDP message is 142 byte = 1136 bit. The theoretical total delay is the sum of processing delay, transmission delay, propagation delay, and

queuing delay. The transmission rate with a 1000 Mbps Ethernet connection is 1 bit/ns. Processing delay can be neglected since the measurement script uses hardware timestamps from the time of physical transmitting and receiving a packet. The queuing delay can be assumed as minimal since only one single message will be sent on the link between sender and receiver at the same time. With a packet size of 142 byte the transmission delay would be 1136 ns. Under the assumption that data can be transmitted on a copper cable with 2/3 of the speed of light, the corresponding propagation delay of a 7.5 m cable is 37.5 ns. This leads to a theoretical total delay of 1173.5 ns.

Figure 4.5 shows the comparison between a *valid* and an *invalid* one-way latency measurement using hardware timestamps. The impact on the mean path delay is minimal with 15 ns offset. But the difference in standard deviation (924 ns) is significant. During the uncalibrated state, every clock is in a *free-running* mode due to the missing synchronization. This leads to drifting clocks and therefore a wider deviation of latency. Even when the state change back to a *locked state*, a small *warm-up* phase is recognizable. Table 3.12 shows this warm-up period and the correlated *master offset* values. Since every measured dataset has the associated Linuxptp values stored, the *valid* dataset only use data after the *warm-up* phase. This leads to a smaller standard deviation compared to *invalid* dataset.

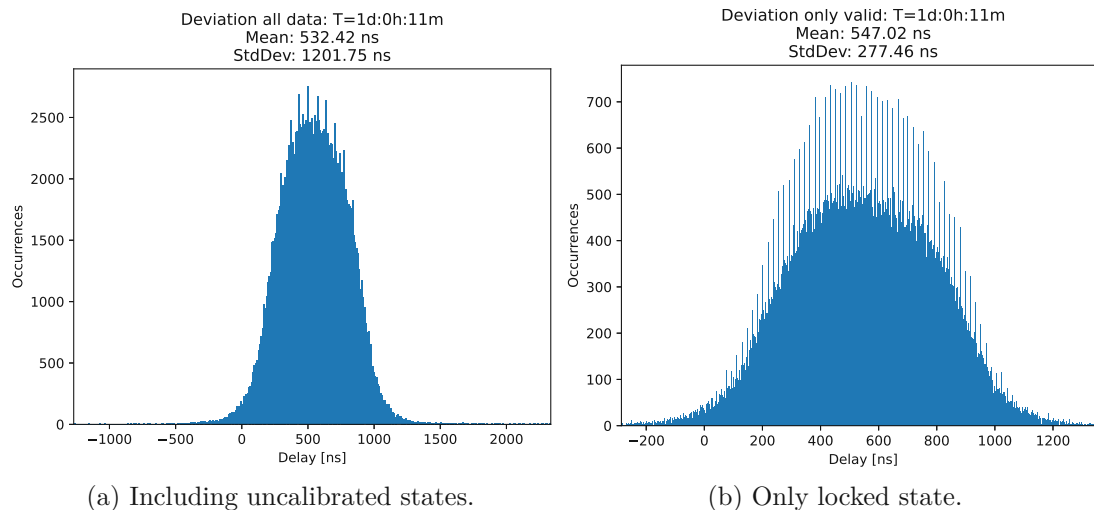


Figure 4.5: One-way latency of direct Ethernet connection, influenced of uncalibrated state using hardware timestamps.

There are multiple reasons to explain the 589 ns difference between theoretical and measured path delay. 100% clock synchronization is not possible, so the individual synchronized clocks of the sender and receiver are also affecting the latency. The predominant part of the theoretical path delay is the transmission delay with 1136 ns. Per definition, this is the time that the physical layer needs to push the packet on the link [47]. As explained in Chapter 4.1.2 transmitted packets are looped back to the

MSG_ERRQUEUE to timestamp the actual time of transmitting the packet. Therefore it is plausible that the measured path delay is smaller than the calculated total delay of 1173.5 ns.

One benefit of the developed clock synchronization concept is the hardware timestamp feature of the Intel i350 NIC, this allows accuracy in nanosecond resolution. To demonstrate this major advantage, the same measurement is repeated with a UDP socket configured for software timestamps (see Chapter 4.1.1). Figure 4.6 illustrates the mentioned drawbacks of software timestamps. Both datasets, *valid* and *invalid*, show two peaks with an offset of $4.2 \mu\text{s}$ instead of a normal distribution compared to hardware timestamps. According to the data, it is not a specific amount of consecutive packets that are delayed. Instead, time stamping is delayed randomly from other preemptive tasks. Another aspect is the similar standard deviation between *valid* and *invalid* dataset. This can be explained due to less accurate timestamps in general. As explained in Figure 4.5, the influence of the short warm-up phase after a state switch is sub-microsecond. So the resolution of software timestamps is too low to significantly influence the path delay measurement.

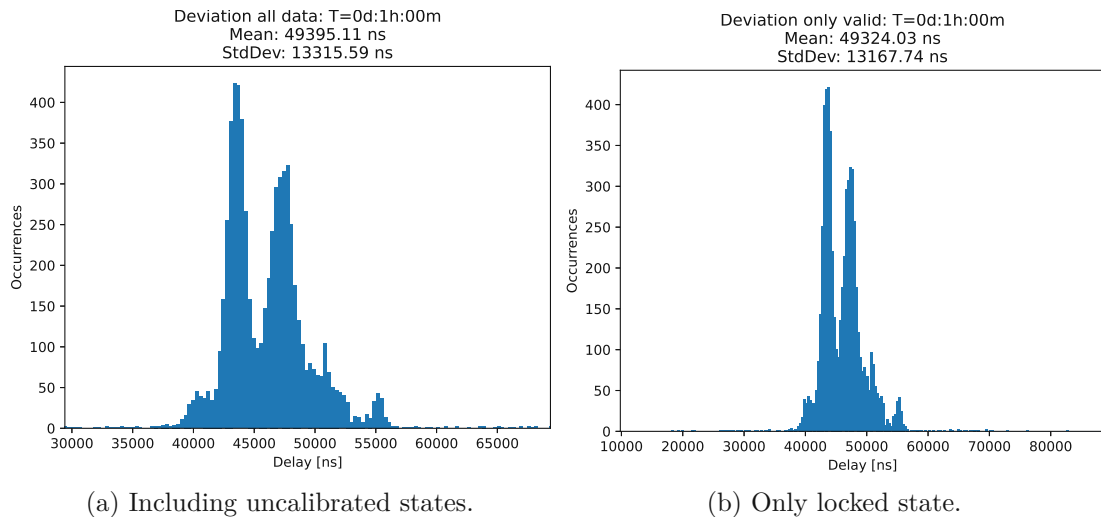


Figure 4.6: One-way latency of direct Ethernet connection, influenced of uncalibrated state using software timestamps.

To further prove the reliability of the developed UDP measurement script, results from Figure 4.6 can be compared with RTT values from a *ping* command. As explained in Chapter 4.1.1 ping command is also using similar software timestamp socket configurations. The average RTT of a ping command with the same packets size (142 Byte) over the same physical Ethernet connection is $182 \mu\text{s}$. This leads to an offset of $83 \mu\text{s}$, between ping RTT and double the time of measured $49.4 \mu\text{s}$ one-way latency. This offset is plausible since RTT also includes processing time within the NIC to prepare the ping response.

Another reference measured during this setup are timestamps from the `clock_gettime()` method including current time from `CLOCK_REALTIME` and

`CLOCK_MONOTONIC_RAW` (see Chapter 4.1.2). This method is called right before and after sending/receiving a UDP packet (see Listing 4.3). It is unpredictable how many preemptive tasks might be executed between this and `sendmsg()/recvmsg()` method call. Therefore, this method is not recommended for precise delay measurements seeking sub-microsecond resolution. Figure 4.7a shows the absolute clock drift between two not synchronized systems. The current clock offset and absolute drift between two not synchronized clocks can be calculated with Equation 4.1a and 4.1b.

$$clock_offset = (RX_{ts} - TX_{ts}) - path_delay \quad (4.1a)$$

$$absolute_drift = clock_offset[end] - clock_offset[start] \quad (4.1b)$$

The `path_delay` value is used from the hardware timestamp measurement and `RX/TXts` represent the current `CLOCK_MONOTONIC_RAW` value. This leads to an absolute drift of $114\ \mu\text{s}$ within one hour between two not synchronized computers. This is a rare scenario but should demonstrate the necessity of synchronization concepts. Figure 4.7b shows the measured path delay only with `clock_gettime(CLOCK_REALTIME)` method timestamps. Similar to the software timestamps from Figure 4.6 is this user space timestamp affected by other preemptive tasks that lead to this two peaks. The

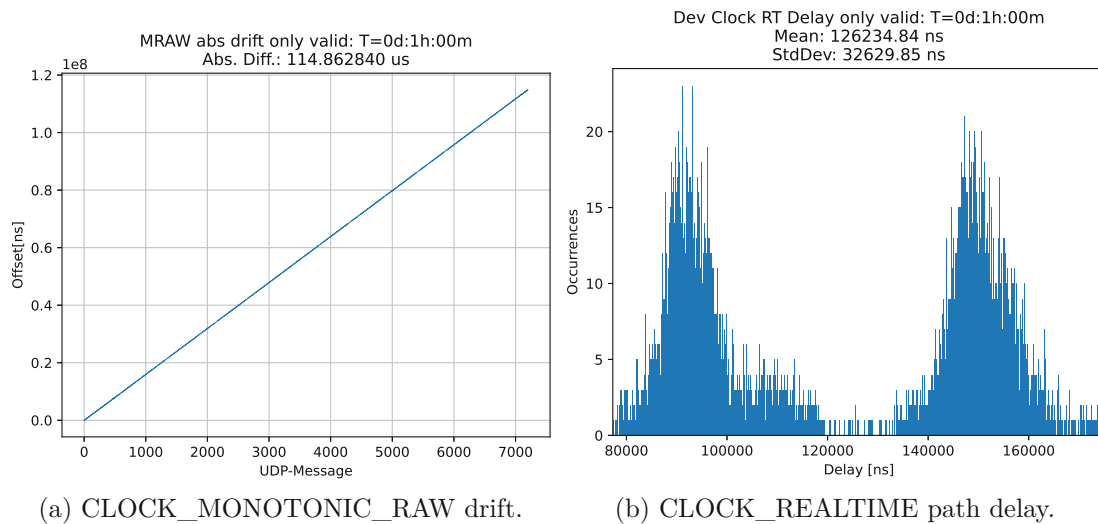


Figure 4.7: Examples of possible clock drift, or path delay measurement accuracy without PTP synchronization.

demonstrated results can prove the reliability of the developed UDP measurement script and the capability of one-way path delay measurement with nanosecond resolution. To reduce the number of redundant plots and results the following sections will only include data from the *valid* datasets. Moreover, reference path delay results from software timestamps, `CLOCK_REALTIME`, or `MONOTONIC_RAW` will be only discussed if they are relevant. Finally, Table 4.1 provides a brief summary of the obtained one-way path delays (except the ping command is RTT).

Table 4.1: Comparison of path delay results depending on measurement method.

Packet Size: Method	142 Byte Path Delay
Hardware timestamp	546.5 ns
Softwaretimestamp	49 300.0 ns
ping (RTT)	182 000.0 ns
clock_gettime()	163 900.0 ns

Real Hardware Switch

Since the correlation between *valid* and *invalid* data, hardware and software timestamps is similar to the previous direct measurement, only results from hardware timestamps are discussed in this section. The Trendnet TE100-S16R is a 100 Mbps switch where EdgeSwitch 12 and Netgear M4300-52G support 1000 Mbps. The cable length from Optiplex 9010 to the switch is 15 m, and the 7.6 m to the PowerEdge R640. Figure 4.8 shows the path delay between the two Dell computers over three different hardware switches. The Trendnet switch has an average latency of 18.471 μ s, where EdgeSwitch has 6.385 μ s and Netgear 5.336 μ s. The theoretical propagation delay of a 7.5 and 15 m cable is neglectable compared to the long processing delay within the switch.

For a quick sanity check, the Forwarding Rate of a 100 and a 1000 Mbps switch can be used to calculate the estimated processing time (see Equation 4.2) [48].

$$Packet_Forwarding_Rate = Transmission_Rate / 8 \text{ bit} / (64 + 8 + 12) \text{ byte} \quad (4.2)$$

$$PFR_{100Mbps} = 100 * 1e6 / 8 / 84 = 148\,809.5 \text{ pps}$$

$$PFR_{1000Mbps} = 1\,488\,095 \text{ pps}$$

The minimal packet length is 64 byte with additional 8 byte frame header and 12 byte frame interval. Thus, the theoretical switch processing delay for a 64 byte message is 1/PFR seconds. For small packet sizes of 142 byte like in this case, a linear correlation between packet size and processing delay can be assumed. Therefore, the calculated delay of the 100 Mbps switch is 14.9 μ s compared to the measured 18.4 μ s. The measured one-way path delay from all three hardware switches are also available in Table 4.2. In addition, the measured latency of the same Netgear hardware switch from [6, Figure 6] is listed in the Table. This demonstrates the improved accuracy of path delay measurement due to the developed PTP Backbone, compared to the measurement introduced in Ogilvie et al. [6].

Emulated Switch in CORE

In this scenario, the same cable length configuration is used as in the previous hardware switch measurement. It can be assumed that an emulated software base switch has a higher latency than a real hardware switch. Therefore, as initial reference, the configurable link delay value of an emulated switch is set to zero.

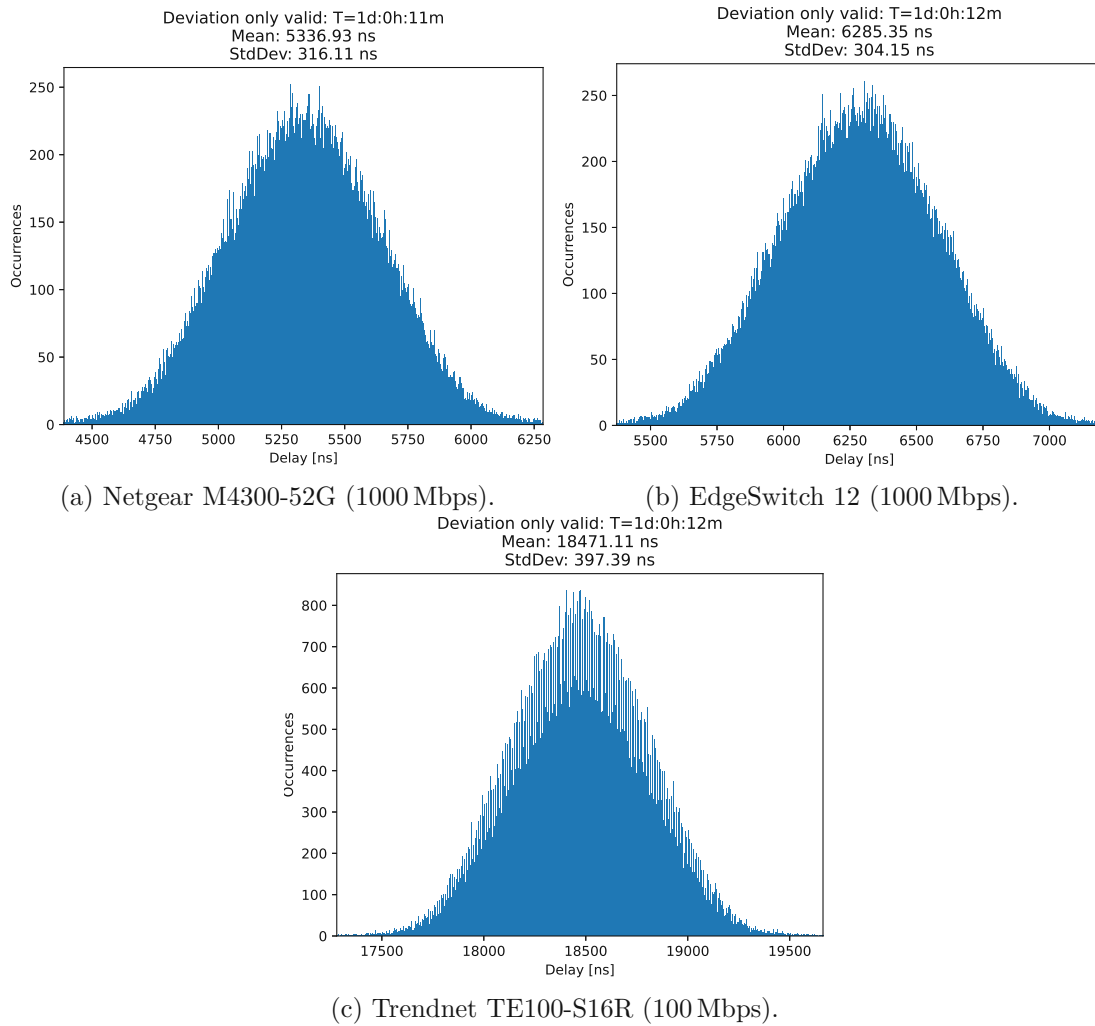


Figure 4.8: Comparison of one-way path delay over three different hardware switches.

Table 4.2: Comparison of one-way path delay over three different hardware switches.

Packet Size:	142 Byte	
Method	Path Delay	Ref. Paper [6]
Trendnet TE100-S16R (100 Mbps)	18.471 μ s	153 μ s
EdgeSwitch 12 (1000 Mbps)	6.285 μ s	
Netgear M4300-52G (1000 Mbps)	5.336 μ s	

In the first phase, the links between the external interface port and the emulated switch are configured with a 1000 Mbps Bandwidth but zero additional delays (see Figure 4.4). In the second phase, different virtual link delays are configured to compare the added delay with the real measurement (see Figure 4.9 a-c). The same procedure is repeated for a 100 Mbps Bandwidth link (see Figure 4.9d-f). According to Figure 4.9 an emulated 100 Mbps switch behaves similarly to an 1000 Mbps switch regarding path delay with an equal delay value. This could be explained by the low traffic during the measurement. The average minimum achievable path delay in CORE is therefore $65.3 \mu\text{s}$. The accuracy of the emulated additional delay of 50 and $100 \mu\text{s}$ is matching more than 91 % with the measured additional delay.

Table 4.3: Average propagation delay of an emulated switch according to different network parameters.

Emulated Delay [μs]	100 Mbps Link		
	Path-Delay [μs]	Offset to 0 μs [μs]	Data conformity [%]
0	65.189	0	0
50	119.59	54.401	91.2
100	169.952	104.763	95.2

Emulated Delay [μs]	1000 Mbps Link		
	Path-Delay [μs]	Offset to 0 μs [μs]	Data conformity [%]
0	65.299	0	0
50	119.563	54.264	91.5
100	169.703	104.404	95.6

Since the cable configuration is identical to the real hardware switch measurement, comparing data from real and emulated switch will lead to a comparison of the actual processing delay of both concepts. This allows an accurate validation of the emulated switch in CORE. By subtracting the measured path delay from a real and an emulated switch, the additional latency can be validated. Table 4.4 shows that an emulated 100 Mbps switch is $46.72 \mu\text{s}$ slower than the Trendnet switch. And an emulated 1000 Mbps switch is $\sim 59 \mu\text{s}$ slower than EdgeSwitch/Netgear switch.

Table 4.4: Difference between the real and emulated switch in CORE.

	CORE		Ref. Paper [6] 1000 Mbps
	100 Mbps	1000 Mbps	
Trendnet	$46.72 \mu\text{s}$	-	-
EdgeSwitch	-	$59.02 \mu\text{s}$	-
Netgear	-	$59.96 \mu\text{s}$	$-29 \mu\text{s}$

In Ogilvie et al. [6], the presented ping RTT response time from the same Netgear switch and CORE revealed that the response time from CORE was $29 \mu\text{s}$ shorter than the real hardware switch. That specific use case demonstrated that the PTP Backbone can be

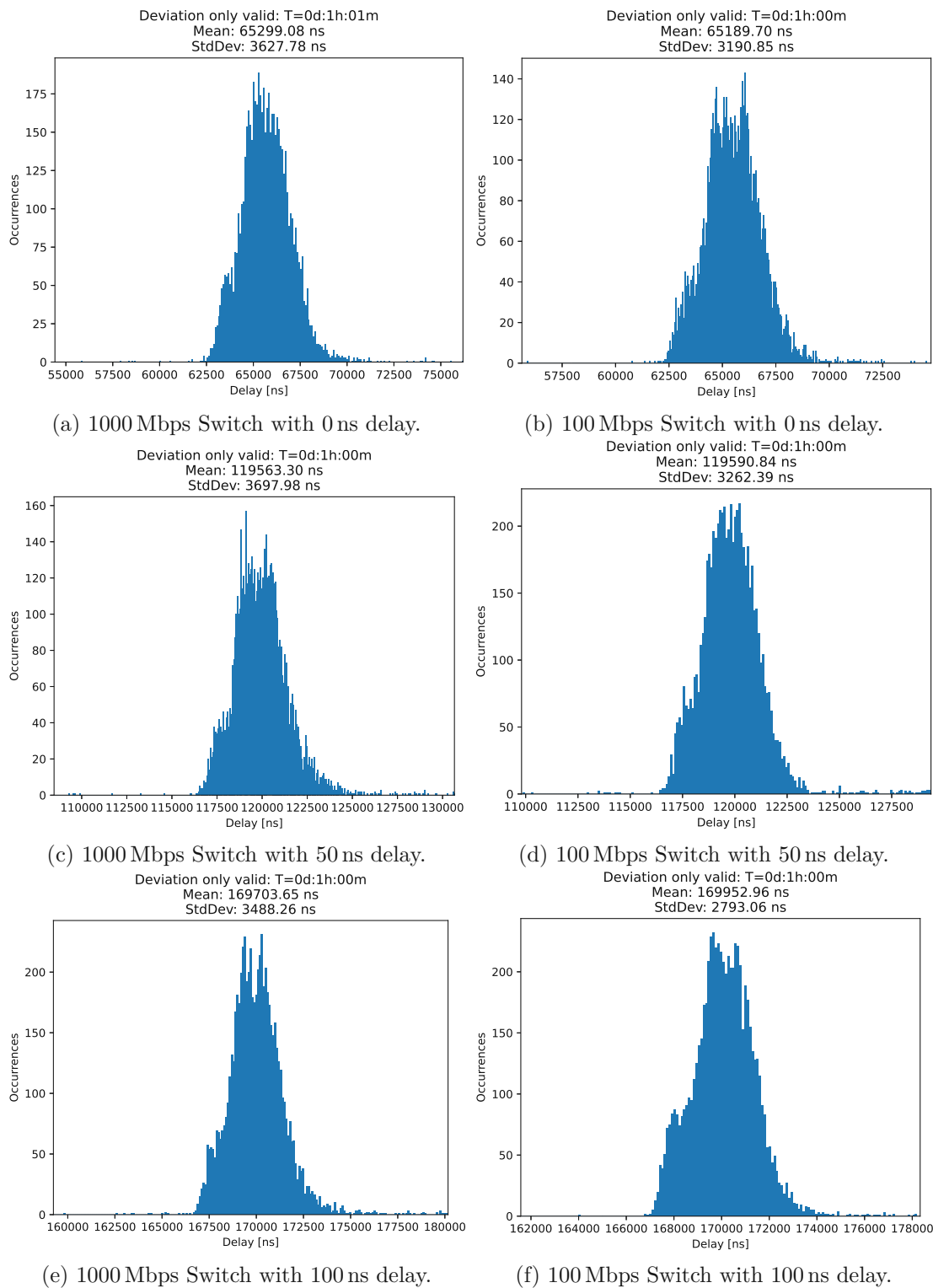


Figure 4.9: Deviation of the propagation delay of an emulated switch according to different network parameters.

used for precise path delay measurements compared to the measurement method of the reference work [6].

4.3 RTDS Validation Process

4.3.1 RTDS Measurement Setup

The GTNET card is a real-time communication interface between the simulation and an external network. It supports a huge variety of protocols depending on the application. For this scenario, the GTNET-SKT is relevant to exchange data via UDP. Figure 4.10 shows the Draft file within RSCAD, including all necessary modules to implement a UDP communication with synchronized timestamps from the GTSYNC card.

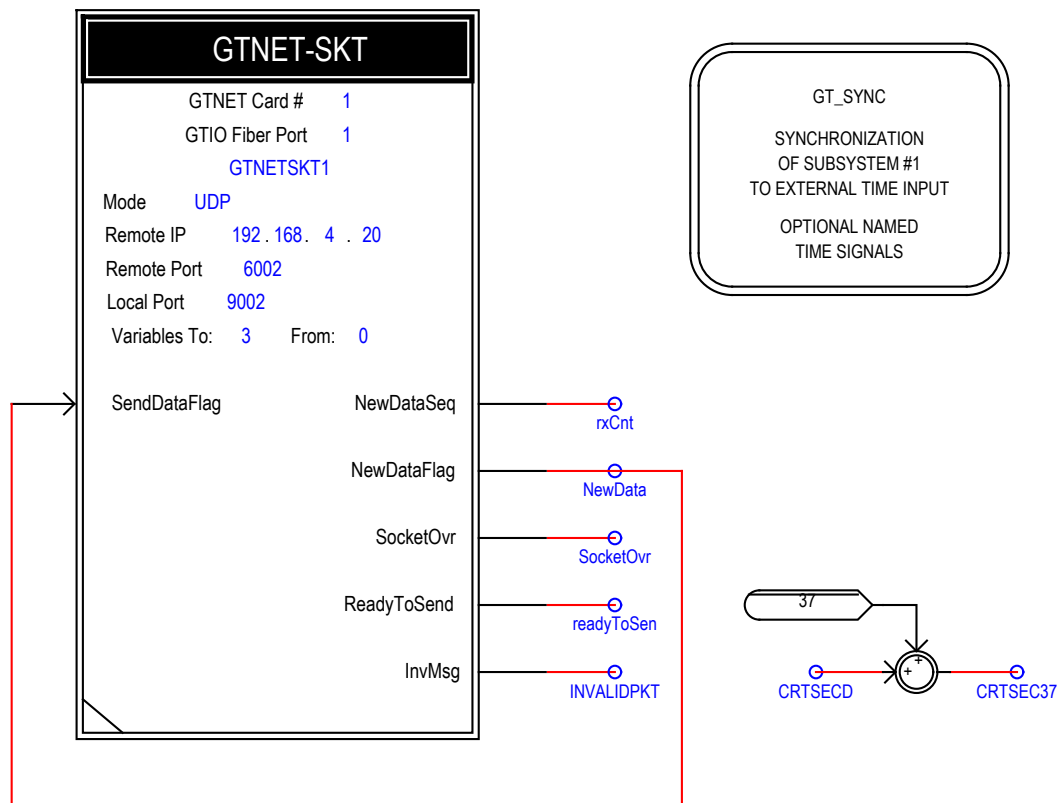


Figure 4.10: Schematic of the RSCAD UDP communication using the GTSYNC card as a time reference.

By importing the GT_SYNC module, two important time signals are available, *CRTSECD* (current second) and *CRTNSECD* (current nanosecond) since Epoch. As mentioned in Chapter 3.2.3 only TAI time format is available in IEC 61850-9-3 mode, but the GT-

SYNC card expects UTC. Therefore, the 37s offset between both is added manually and the new signal is labeled as *CRTSEC37*. In general, only the GT_SYNC module in combination with the 37s offset are necessary to add PTP synchronized timestamps to any simulation case.

Besides the network configuration like IP address and port from the GTNET-SKT, it is required to set the *from:* variable properly. The GTNET card expects data length with integer multiples of four byte. Since the transmitted packets have a data length of 100 byte, the variable has to be set to 25.

Several other flags are relevant. *SocketOvr* is an Integer output signal that indicates a loss of UDP packets. This can occur if multiple messages arrive at the same time. If an Ethernet message with the wrong length is received the *InvMsg* would be set to '1'. *ReadyToSend* is only '1' if port configuration is completed and no pending data transmission exists. Only then, GTNET-SKT is ready to receive data from the processor (like timestamps) that will be used as UDP payload.

The Integer value of the *NewDataSeq* flag increments for every received message since the simulation started. This value can be used to add a unique consecutive Integer number to every transmitted UDP message. In addition, the payload includes the current timestamp from the GT_SYNC module in seconds and nanoseconds (*CRTSEC37* and *CRTNSEC*).

If the *SendDataFlag* input is set to '1' and the *ReadyToSend* flag is also '1', two timesteps are required to send the data. In the first timestep, the dataset including the GTSYNC timestamps, are sampled from the processor. In the second timestep up to 30 data points are transferred from the processor to the GTNET card. The time for one timestep can be configured in RSCAD in the draft file. By right clicking anywhere in the *Draft* area of RSCAD, under section "Circuit Options", Time Step (μs) parameter is available. The minimum allowed value, regulated by RSCAD, is $10\ \mu\text{s}$. Therefore by connecting the *NewData* signal direct to the to the *SendDataFlag*, receiving a message is also the trigger to send a message back [49]. Figure 4.11 shows the run-time environment of the UDP measurement in RSCAD including all explained signals for monitoring reasons.

Figure 4.12 demonstrates the fundamental concept to measure one-way latency between Linux machine and RTDS. One dataset of n UDP message consist of three different timestamps to calculate the one-way latency for each direction. First, the hardware timestamp from the point of transmitting the UDP message from the Linux machine. Second, the current timestamps from the GT_SYNC module from the moment of receiving a packet. Third, the hardware timestamp from the received message transmitted from the GTNET card, including the GTSYNC timestamps in the message payload. With these three timestamps and the correlated message number n , the path delay for each direction can be calculated.

A small example, in case the timestep is set to $10\ \mu\text{s}$, and a UDP message is received. The *rxCnt* signal would increase by one and *NewData* and thereby also *SendDataFlag* is set to '1'. During the first $10\ \mu\text{s}$ timestep the processor is preparing a dataset, including the value of *CRTSEC37* and *CRTNSEC* from the GTSYNC module. Within another

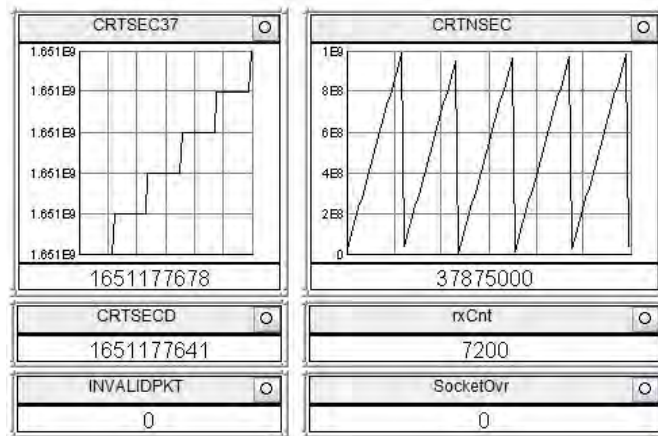


Figure 4.11: Visualization of an example GTNET UDP measurement including all monitoring signals.

10 μ s timestep the data is sent from the processor to the GTNET card and further to the Linux machine via UDP. So in theory, increasing the timestep by 100 μ s should increase the latency between GTNET and Linux machine by 200 μ s. Chapter 4.3.2 will show the latency depending on different timesteps of RTDS.

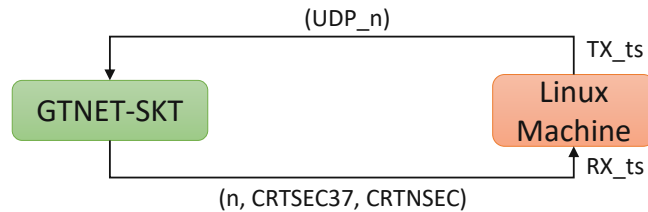


Figure 4.12: Example latency measurement setup between GTNET-SKT and Linux machine.

4.3.2 RTDS Validation Results

The following section will demonstrate the availability of timestamps during a RSCAD simulation that is synchronized to the PTP Backbone through the GTSYNC card. Using the timestamps from the GTSYNC module to measure the latency between a device and RTDS via UDP messages is only one representative way. The results presented in this section also display the impact off different processing timesteps of the RTDS processor. As explained in Chapter 4.3.1 changing the timestep of RSCAD should have a direct impact on the measured latency between Optiplex 9010 and RTDS. Minimal two timesteps are necessary to compute and respond after an incoming UDP message at the GTNET card. To demonstrate the ideal scenario only data and results from the *valid*

dataset will be discussed. Differences between *valid* and *invalid* datasets were discussed already in previous sections.

The minimal timestep within RSCAD that can be set is $10\ \mu\text{s}$. In addition measurements with a timestep of 25, 50, 75, 100, and $150\ \mu\text{s}$ are accomplished. Three representative dataset are presented in Figure 4.13. Figure 4.13a and 4.13b show the influence of path delay depending on the RTDS timestep. Two timesteps are required between requesting GTSYNC current timestamps and actually responding. Thus, the path delay is not symmetric, *to_GTNET* is with $50.1\ \mu\text{s}$ shorter than *from_GTNET* ($79.8\ \mu\text{s}$). The sum of both path delays (see Figure 4.13) correlates with double the time of the set RTDS processor timestep. Increasing the timestep from $25\ \mu\text{s}$ to $50\ \mu\text{s}$, increases the total path delay by $49.665\ \mu\text{s}$. Table 4.5 illustrates the correlation of the other timesteps. The correlation between timestep and path delay of *from_GTNET* is linear between 25 and $150\ \mu\text{s}$ with less than 1% deviation.

Another interesting aspect can be seen in Figure 4.13 a, c, and e. By increasing the timestep the standard deviation of the *to_GTNET* latency is increasing. The right maximum is almost constant at $50\ \mu\text{s}$ but the minimum path delay is getting negative. This phenomenon can be explained in Figure 4.14. At the beginning and the end of every timestep period the RTDS processor is reading and writing data depending on the simulation needs. Depending on the moment of receiving a UDP message and requesting GTSYNC timestamps, the RTDS processor decides whether he takes data from the beginning or obtains new/current values. So the total width of the *to_GTNET* path delay deviation matches with the selected timestep value. By further increasing this timestep, the width of that time window might be larger than the time event of the original sent UDP message from the Optiplex machine. This causes a temporally smaller RX_timestamp than TX_timestamp which leads to a negative path delays. That means that a negative latency value can not be larger than one timestep.

Table 4.5: Path-delay between GTNET and Optiplex machine, depending on RTDS timestep.

Time-Step [ns]	to GTNET	from GTNET	Diff. between Time-Steps	
	Path-Delay [ns]	Path-Delay [ns]	[ns]	[%]
10	50,118	79,757	0,000	0,000
25	42,387	97,345	17,588	70,571
50	29,583	147,010	49,665	0,675
75	18,303	197,177	50,167	-0,333
100	5,193	246,974	49,797	0,408
150	-20,242	346,915	99,941	0,059

Due to the developed PTP backbone and thus the possibility to perform precise one-way latency measurements, it was possible to analyze the behavior of RTDS with respect to the use of timestamps during running simulations.

4. SYSTEM PERFORMANCE VALIDATION

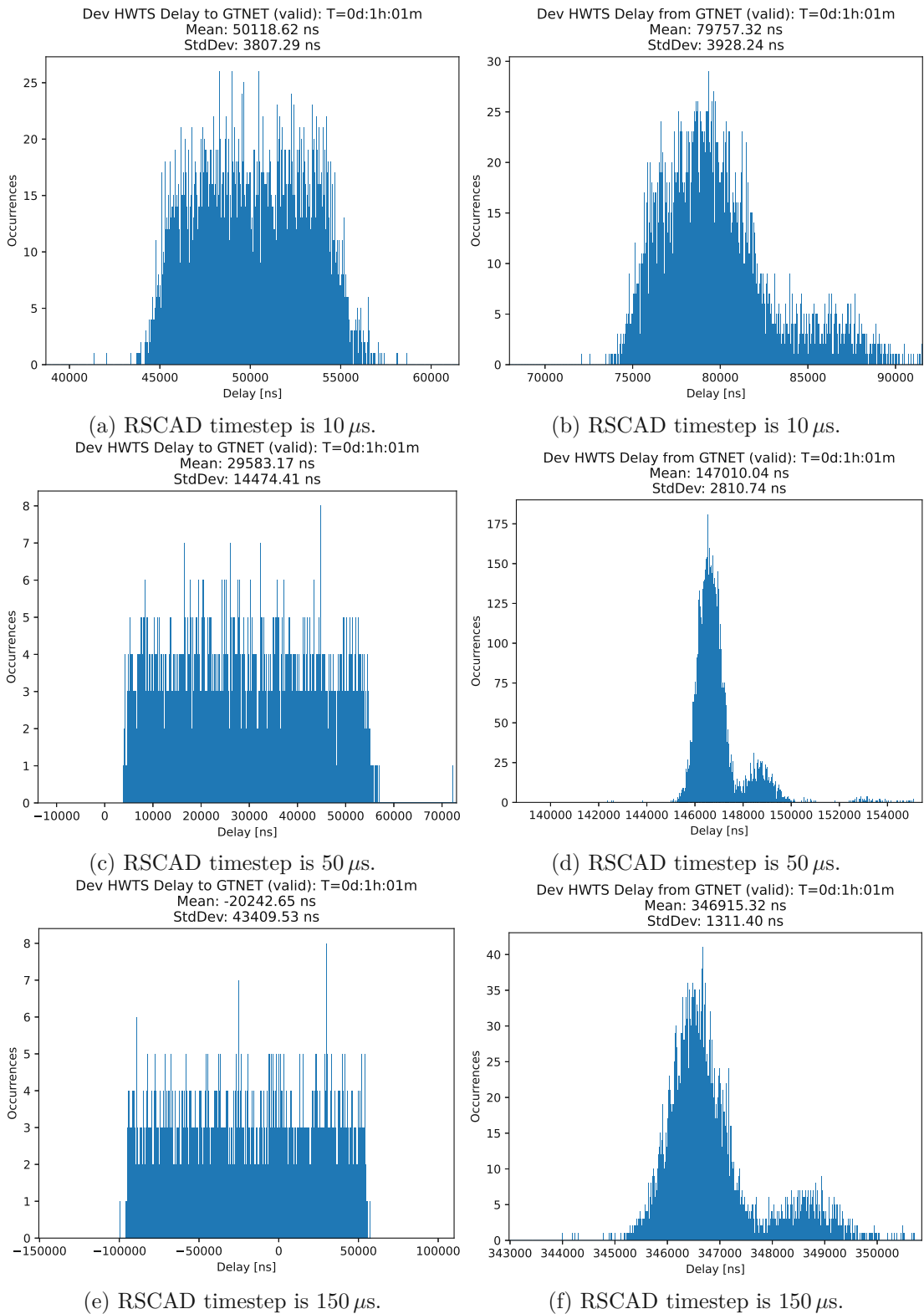


Figure 4.13: Deviation of path delay between Optiplex 9010 and RTDS.

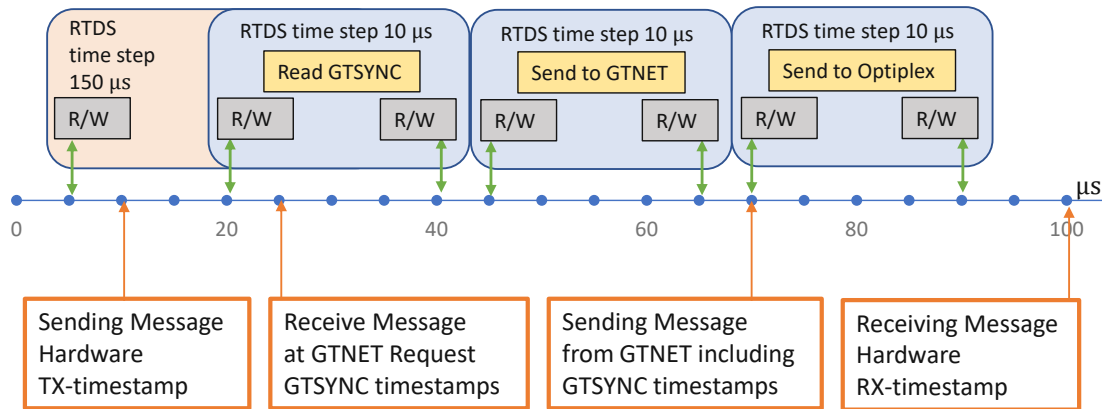


Figure 4.14: Influence of the timestep on the measured path delay.

4.4 SyncBox validation Process

4.4.1 SyncBox Measurement Setup

During HIL simulations scopes are relevant for long term monitoring or capturing a current state. In both scenarios the scope should be placed as close as possible to the DUT. This is common practice since it reduces unnecessary long analog coax cables and the latency between measured events and their temporal data acquisition. As explained in Chapter 3 scopes can be best effort synchronized to the PTP Backbone via the Meinberg SyncBox. Therefore, during larger or distributed simulations, multiple SyncBoxes required to synchronize devices with a 10 MHz reference input with the PTP Backbone. Thus, it is important to verify the accuracy of clock stability between the 10 MHz reference output signal of the individual SyncBoxes.

Figure 4.15 illustrates the setup to measure the clock stability of two Meinberg SyncBoxes. Both devices are in sync with the PTP Backbone, and internal oscillators are warmed up. The third output port from SyncBox 1 is used for the 10 MHz reference input of the Tektronix MDO4034C scope. The port configuration between scope and both SyncBoxes is visible in Figure 4.15. Two major parameters are interesting, the internal clock stability within one SyncBox, and between two separated SyncBoxes. According to the datasheet the accuracy of a pulse output synchronized to PTP is ± 100 ns (relative to the GMC) and ± 1 s with NTP [38].

During the data post-processing in Python several steps are necessary to verify the clock stability. The scope is capturing the four 10 MHz rectangular signals over a period of one day and stored it to mat-files in HDF5 format. Every file contains data from a 0.008 s period. This results in 80 000 rising edges per signal. Channel 1 (Output 1 from SyncBox 1) is defined as reference. To quantify the clock stability between Channel 1 and every other Channel, the offset between each rising edge can be calculated. The minimum, average and maximum offset of each mat-file are saved in a data-frame. The

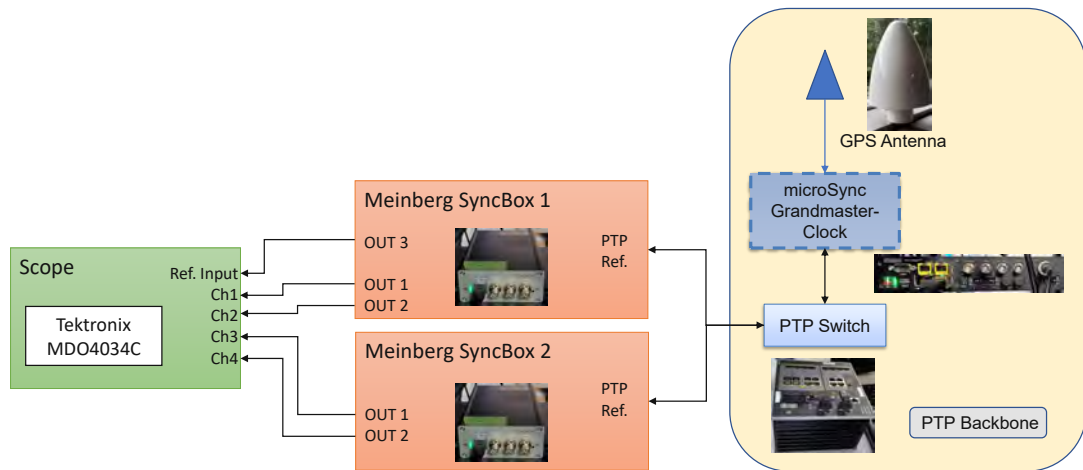


Figure 4.15: Measurement setup for SyncBox clock stability.

average time between each scope capture is two minutes, so a possible long-term drift can be measured.

4.4.2 SyncBox Validation Results

This section will present the clock accuracy of the 10 MHz output signals between two SyncBoxes. A detailed measurement setup description was presented in Chapter 4.4.1 and Figure 4.15. Figure 4.16 shows an example of the rising edge detecting algorithm. According to the signal frequency of 10 MHz and a span of 0.008 s per scope capture, a theoretical number of 8000 rising edges results. The used algorithm was able to capture all 8000 rising edges.

Figure 4.17 illustrates the minimum and maximum offset between a rising edge of reference Channel 1 (OUT1 of SyncBox 1) and every other SyncBox output. During a span of 0.008 s Channel 2 has a offset of 1.4 ns and drifts ± 0.2 ns compared to Channel 1. Channel 3 and 4 showed a constant offset of -12.6 ns and -12.4 ns, and a drift of ± 0.4 ns. This higher offset is reasonable since Channel 3 and 4 are output signals of the second SyncBox therefore the offset is correlated to the PTP synchronization accuracy. These three values, average, minimal, and maximal offset of each 0.008 s measured period, are captured 1551 times over a total measured time of ~ 41 hours.

Figure 4.18 shows the minimum, average, and maximal drift compared to Channel 1 during 40 min period of the measurement. Since the average offset correlates with the minimum and maximum curve with less than 0.7 ns, only average values are further discussed. Therefore, Figure 4.19 illustrates the minimum and maximum drift compared to Channel 1 during the total 41 hours measurement duration. The drift within SyncBox 1 (Channel 2 to Channel 1) is expected small with maximum value of 1.37 ns. SyncBox 2 is drifting between -16.1 and 84 ns compared to reference Channel 1. According to the

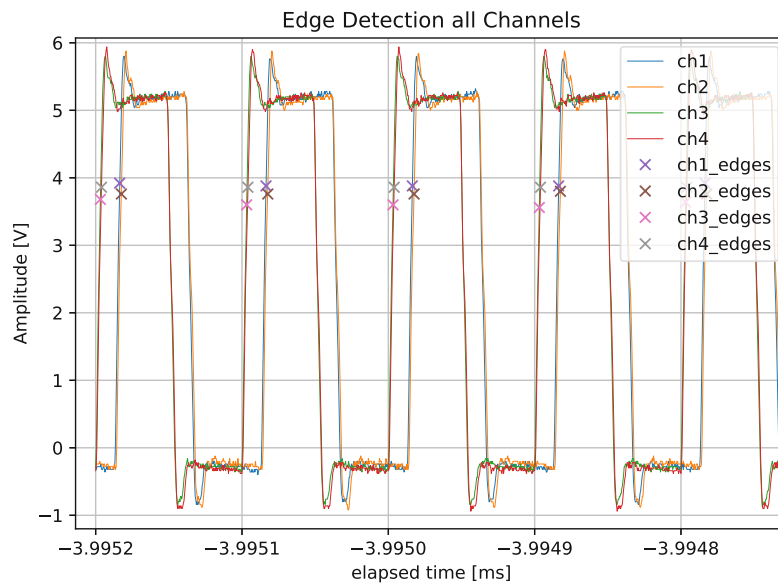


Figure 4.16: Example of the rising edge detection algorithm.

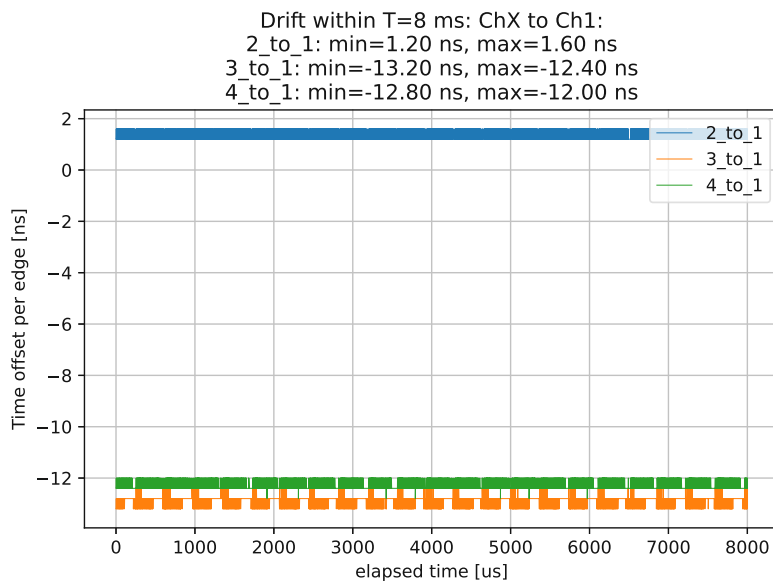


Figure 4.17: Min/Avg/Max clock drift compared to reference Channel 1.

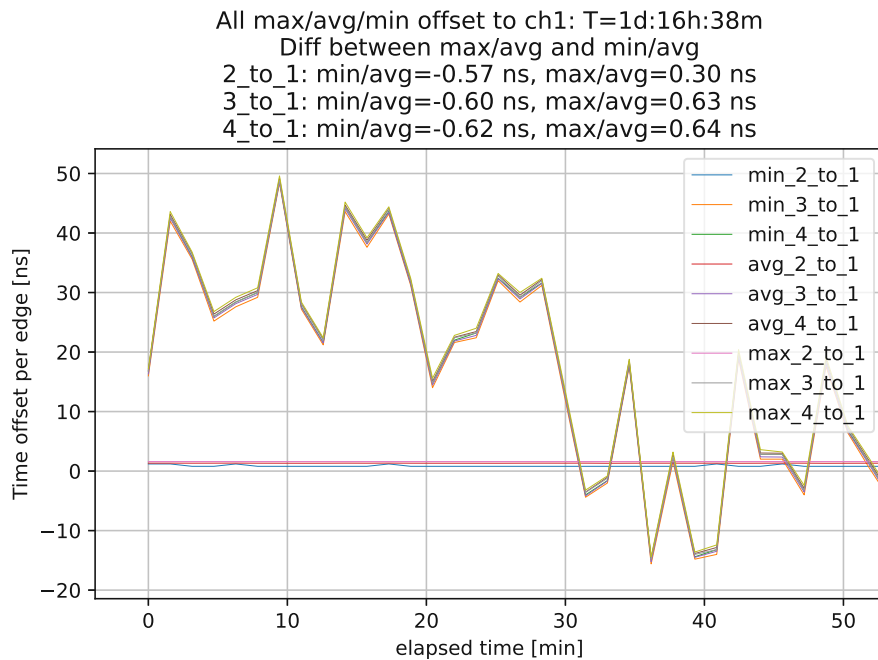


Figure 4.18: Minimal, average, and maximal drift compared to Channel 1 during 40 min period.

datasheet, the accuracy of the pulse output synchronized to PTP is ± 100 ns (relative to the GMC). Thus, the drift between the two SyncBoxes is within the tolerance.

Figure 4.20 compares the drift between two 10 MHz outputs of the same SyncBox. Since the Tektronix MDO4034C scope only has four input channels, it is not possible to compare all six output channels of both SyncBoxes. But it can be assumed that the behavior of the third channel can be derived from the drift between the other Channels. The maximum internal drift of SyncBox 2 (*avg_4_to_3*) is 0.6 ns smaller than SyncBox 1. In Figure 4.20a nine negative spikes occur during a 41 hour measurement. Figure 4.20b shows a specific spike at 862 min in detail. Internal clocks and thus external driver is synchronized to the external PTP Backbone. It seems like one output driver is getting the information of the new value, one timestep before the other channel. Table 4.6 summaries the relevant measurement results again. Since the minimum internal drift of the SyncBox 2 is -99.3 ns, due to the peaks, the average value is more applicable.

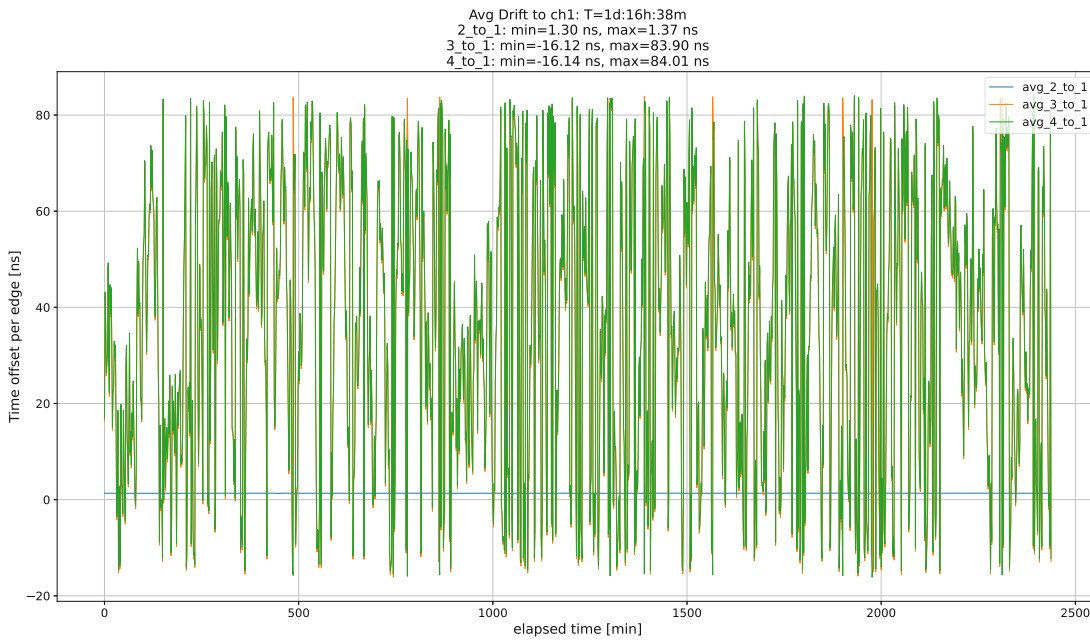
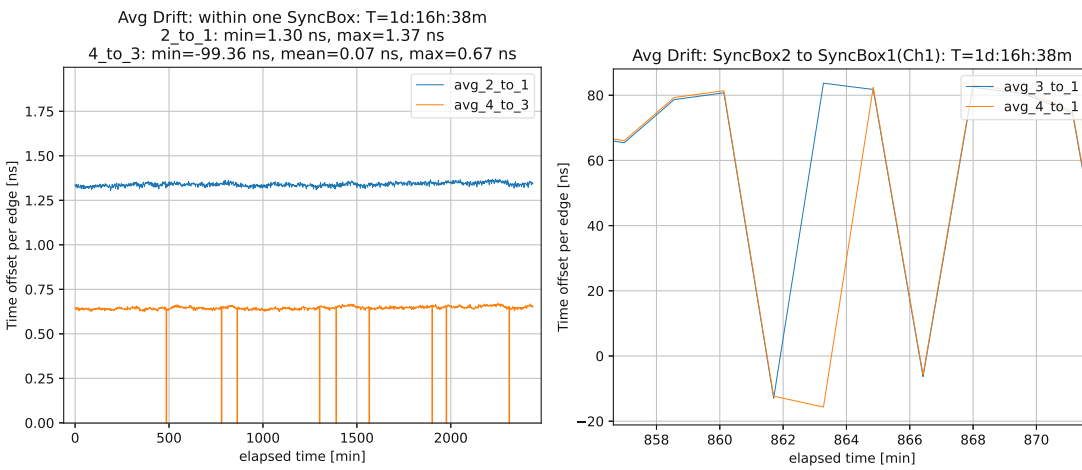


Figure 4.19: Average drift compared to Channel 1 during the total measurement period.



(a) RSCAD timestep is $10 \mu\text{s}$.

(b) RSCAD timestep is $10 \mu\text{s}$.

Figure 4.20: Drift between 10 MHz output ports within the SyncBox.

Table 4.6: Summary of internal and external drifts of two SyncBoxes.

External	10MHz Output Drift		
	min [ns]	avg [ns]	max [ns]
Ch2 to Ch1	1,305	-	1,37
Ch3 to Ch1	-16,123		83,9
Ch4 to Ch1	-16,139	-	84,008
Internal			
Ch2 to Ch1	1,305	-	1,37
Ch4 to Ch3	-99,36	0,066	0,673

Outlook

5.1 Conclusion

This work has demonstrated the possibility to integrate a GPS synchronized PTP Backbone into an existing HIL testbed. Devices with hardware timestamp support and a PTP implementation can benefit from the common global GPS time reference. A RTS like RTDS can use its GTSYNC card to synchronize with the PTP network. Thus, synchronized timestamps are accessible during real-time simulation. Machines with Linux OS and hardware timestamp support can be synchronized due to the open-source Linuxptp package. The Meinberg SyncBox has the PTP implemented and provides several time sources like 10 MHz, 1PPS or IRIG time codes. This enables even devices without PTP implementations the capability to synchronize their clocks with the common GPS time reference.

One representative user application to demonstrate the benefits of the implemented PTP Backbone is the possibility to proceed with one-way path delay measurements. Thus, new knowledge in terms of CORE performance and the impact of timestep size during real-time simulations could be obtained. In previous work from Ogilvie et al. [6], the measured latency of an emulated switch within CORE was $124\ \mu\text{s}$ compared to a real hardware switch with $153\ \mu\text{s}$. Due to hardware timestamps and one-way latency measurements presented in this work, latency measurement of the same hardware switch led to $5.3\ \mu\text{s}$ and $65.3\ \mu\text{s}$ for an emulated switch within CORE. CORE has the opportunity to add additional virtual delay to an emulated network. The percentage error between configured and actual measured delay could be validated as below 10%.

Another new insight is the impact of the real-time simulation timestep size on timestamp accuracy. The deviation of latency measurements between the RTS from RTDS correlated with timestep length. That means, that the accuracy of timestamps during a real-time simulation is directly correlated to the simulator processor's timestep size. The presented

results and insights are demonstrating the benefits of the PTP Backbone in that particular user application. The design and the precision of the PTP Backbone enable a variety of possible applications in the future.

5.2 Future Work

One major advantage of a GPS synchronized HIL testbed is the opportunity for co-simulation around the world. Therefore, this work could act as guidelines for other organizations to implement such a system in their testbed. Thus, future work could demonstrate the accuracy of HIL co-simulations using the same global time base. Linuxptp drops some event messages during runtime which causes an uncalibrated state for several seconds. During this time, clocks within the PTP network might drift since synchronization is not possible during this state. This can happen, due to other preemptive tasks that cause a significant delay of PTP event messages. In the future, the focus could be set on the priority of PTP event messages of the Linuxptp service to avoid this behavior. Similar to the work from Rinaldi et al.[5] the 1PPS reference signal from the Meinberg SyncBox could be used to synchronize components that do not implement the PTP. For communication protocols like CAN bus an Ethernet-CAN-Gateway FD DR from Peak-System is already purchased as part of the PTP Backbone design concept but is currently not implemented in the testbed. Future work could investigate procedures to timestamp data from a DUT using CAN bus as a communication protocol. Security aspects are not considered during this work since the demonstrated concepts are not part of critical infrastructure. Nevertheless, working with PHIL simulation requires correct timing constraints to achieve a safe and successful simulation. Thus, game theories could be investigated to estimate potential risks of cyber attacks on PTP networks and clock timings.

Bibliography

- [1] HBM. Precision time protocol in data acquisition and testing. [Online] Available: <https://www.hbm.com/en/5143/precision-time-protocol/>. (accessed 22 May 2021).
- [2] Huang Xin, Li Wenmeng, Yang Song, Zhang Daonong, and Du Qiwei. Smart substation iec61588 time synchronization system and security evaluation. In *2014 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 97–101. IEEE, 2014.
- [3] Chris S. Edrington, Michael Steurer, James Langston, Touria El-Mezyani, and Karl Schoder. Role of power hardware in the loop in modeling and simulation for experimentation in power and energy systems. *Proceedings of the IEEE*, 103(12):2401–2409, 2015.
- [4] Oscar Azofeifa, Siddhartha Nigam, Olaoluwapo Ajala, Christopher Sain, Samuel Utomi, Alejandro D. Dominguez-Garcia, and Peter W. Sauer. Controller hardware-in-the-loop testbed for distributed coordination and control architectures. In *2019 North American Power Symposium (NAPS)*, pages 1–6. IEEE, 10/13/2019 - 10/15/2019.
- [5] Stefano Rinaldi, Federico Bonafini, Paolo Ferrari, Alessandra Flammini, Marco Pasetti, and Emiliano Sisinni. Software-based time synchronization for integrating power hardware in the loop emulation in iec61588 power profile testbed. In *2019 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 1–6, [Place of publication not identified], 2019. IEEE.
- [6] Colin Ogilvie, Juan Ospina, Charalambos Konstantinou, Tuyen Vu, Mark Stanovich, Karl Schoder, and Mischa Steurer. Modeling communication networks in a real-time simulation environment for evaluating controls of shipboard power systems. In *2020 IEEE CyberPELS (CyberPELS)*, pages 1–7. IEEE, 2020.
- [7] RTDS. *Chapter 11 GTSYNC*. RSCAD v5.011 (accessed 24 May 2022).
- [8] Richard Cochran. Github · richardcochran/linuxptp. [Online] Available: <https://github.com/richardcochran/linuxptp>. (accessed 20 April 2021).

- [9] Meinberg. Oscillators available for meinberg receivers: Ocxo, tcxo. [Online] Available: https://www.meinbergglobal.com/english/specs/gpsopt.htm?pk_source=print&pk_medium=flyer&pk_campaign=OSC-List&pk_content=&pk_cid=21. (accessed 11 May 2022).
- [10] Meinberg. mycosync rx102. [Online] Available: https://www.meinbergglobal.com/download/docs/manuals/english/microsync_rx102_acdc.pdf. (accessed 10 August 2021).
- [11] Juan Ospina, Charalambos Konstantinou, Mark Stanovich, and Mischa Steurer. Evaluation of communication network models for shipboard power systems. In *2021 IEEE Electric Ship Technologies Symposium (ESTS)*, pages 1–9. IEEE, 2021.
- [12] Francisco Girela-Lopez, Jose Lopez-Jimenez, Miguel Jimenez-Lopez, Rafael Rodriguez, Eduardo Ros, and Javier Diaz. Ieee 1588 high accuracy default profile: Applications and challenges. *IEEE Access*, 8:45211–45220, 2020.
- [13] Elena Lisova, Elisabeth Uhlemann, Wilfried Steiner, Johan Akerberg, and Mats Bjorkman. Game theory applied to secure clock synchronization with ieee 1588. In *ISPCS*, pages 1–6, Piscataway, NJ, 2016. IEEE.
- [14] Geoffrey M. Garner. Use of ieee 1588 best master clock algorithm in ieee 802.1as. [Online] Available: <https://www.ieee802.org/1/files/public/docs2006/as-garner-use-of-bmc-061114.pdf>, 200. (accessed 21 May 2022).
- [15] CISCO. Precision time protocol software configuration guide for ie 4000, ie 4010, and ie 5000 switches. [Online] Available: https://www.cisco.com/c/en/us/td/docs/switches/lan/cisco_ie4000/software/release/15-2_4_e/b_ptp_ie4k.html. (accessed 22 May 2021).
- [16] Geoffrey Garner and Hyunsurk Ryu. Synchronization of audio/video bridging networks using ieee 802.1as. *IEEE Communications Magazine*, 49(2):140–147, 2011.
- [17] R.E. Mackiewicz. Overview of iec 61850 and benefits. In *2006 IEEE PES Power Systems Conference and Exposition*, pages 623–630, 2006.
- [18] Communication networks and systems for power utility automation - part 9-3: Precision time protocol profile for power utility automation, May 2016.
- [19] Siddhartha Nigam, Olaoluwapo Ajala, Alejandro D. Domínguez-García, and Peter W. Sauer. Controller hardware in the loop testing of microgrid secondary frequency control schemes. *Electric Power Systems Research*, 190:106757, 2021.
- [20] Peter J. G. Teunissen and Alfred Kleusberg. Gps observation equations and positioning concepts. In *Lecture Notes in Earth Sciences, volume 60*, pages 175–217, 1996.

- [21] Chang Bok Lee, Dong Doo Lee, Nak Sam Chung, M. Imae, C. Miki, M. Urtsuka, and T. Morikawa. Development of a gps time comparison system and the gps common-view measurements. *IEEE Transactions on Instrumentation and Measurement*, 40(2):216–218, 1991.
- [22] John Klobuchar. Ionospheric time-delay algorithm for single-frequency gps users. *IEEE Transactions on Aerospace and Electronic Systems*, AES-23(3):325–331, 1987.
- [23] D. C. Jefferson, S. M. Lichten, and L. E. Young. A test of precision gps clock synchronization. In *Proceedings of the 1996 IEEE International Frequency Control Symposium (50th anniversary)*, pages 1206–1210, [New York, N.Y.] and Piscataway, N.J., 1996. Institute of Electrical and Electronics Engineers and IEEE Service Center.
- [24] W. I. Bertiger et al. S. M. Lichten, Y. E. Bar-Sever. Gipsy-oasis ii: A high precision gps data processing system and general satellite orbit analysis tooln. In *Technology 2005 NASA Technology Transfer Conference*, [New York, N.Y.] and Piscataway, N.J., 1995-Oct.-24-26.
- [25] coreemu/core - common open research emulator. [Online] Available: <https://github.com/coreemu/core>. (accessed 20 May 2022).
- [26] Jeff Ahrenholz. Comparison of core network emulation platforms. In *Military Communication Conference, 2010*, pages 166–171, [Piscataway, N.J.], 2010. [IEEE].
- [27] E. Guillo-Sansano, M. H. Syed, A. J. Roscoe, G. Burt, Mark Stanovich, and Karl Schoder. Controller hil testing of real-time distributed frequency control for future power systems. In *2016 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pages 1–6. IEEE, 2016.
- [28] James Langston, Kazuki Watanabe, John Hauer, Karl Schoder, Mark Stanovich, Harsha Ravindra, and Michael Steurer. Using power hardware-in-the-loop simulation to study control of energy storage within limited-inertia power system. In *2021 IEEE Electric Ship Technologies Symposium (ESTS)*, pages 1–6. IEEE, 2021.
- [29] Bang L. H. Nguyen, Tuyen Vu, Colin Ogilvie, Harsha Ravindra, Mark Stanovich, Karl Schoder, Michael Steurer, Charalambos Konstantinou, Herbert Ginn, and Christian Schegan. Advanced load shedding for integrated power and energy systems. In *2021 IEEE Electric Ship Technologies Symposium (ESTS)*, pages 1–6. IEEE, 2021.
- [30] S. Suryanarayanan, M. Steurer, S. Woodruff, and R. Meeker. Research perspectives on high-fidelity modeling, simulation and hardware-in-the-loop for electric grid infrastructure hardening. In *2007 IEEE Power Engineering Society General Meeting*, pages 1–4. IEEE, 6/24/2007 - 6/28/2007.
- [31] RTDS. *GTWIF Workstation Interface Card*, March 2012. RSCAD v5.011.

- [32] Richard Cochran, Cristian Marinescu, and Christian Riesch. Synchronizing the linux system time to a ptp hardware clock. In *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 87–92. IEEE, 2011.
- [33] Andras Wiesner and Tamas Kovacs-hazy. Portable, ptp-based clock synchronization implementation for microcontroller-based systems and its performance evaluation. In *2021 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 1–6. IEEE, 2021.
- [34] Canonical. Ubuntu manpage: ptpd - precision time protocol daemon (1588-2008). [Online] Available: <http://manpages.ubuntu.com/manpages/focal/man8/ptpd.8.html>. (accessed 16 August 2021).
- [35] Github - ptpd/ptpd: Ptpd implementation of precision time protocol (ptp). [Online] Available: <https://github.com/ptpd/ptpd>, 2010. (accessed 15 August 2021).
- [36] Maciej Machnikowski, Ramana Reddy, and Zoltan Fodor. Challenges with linuxptp on telco ran deployments. In *2021 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 1–4. IEEE, 2021.
- [37] Tektronix. 5 series mso. [Online] Available: <https://www.tek.com/en/datasheet/5-series-mso>. (accessed 10 August 2021).
- [38] Meinberg. Syncbox/n2x - signal converter. [Online] Available: https://www.meinbergglobal.com/download/docs/manuals/english/syncbox_n2x.pdf. (accessed 3 March 2022).
- [39] Meinberg. Maximum length antenna cable gps-clocks. [Online] Available: <https://www.meinbergglobal.com/english/specs/gpscable.htm>. (accessed 16 April 2021).
- [40] Cisco. Precision time protocol software configuration guide for ie 4000, ie 4010, and ie 5000 switches. [Online] Available: https://www.cisco.com/c/en/us/td/docs/switches/lan/cisco_ie4000/software/release/15-2_4_e/b_ptp_ie4k.html. (accessed 30 July 2021).
- [41] Documentation: time stamping socket programming. [Online] Available: <https://www.kernel.org/doc/Documentation/networking/timestamping.txt>. (accessed 20 September 2021).
- [42] man7.org Michael Kerrisk. socket(2) - linux manual page. [Online] Available: <https://man7.org/linux/man-pages/man2/socket.2.html>. (accessed 5 May 2022).

- [43] man7.org Michael Kerrisk. `setsockopt(3p)` - linux manual page. [Online] Available: <https://man7.org/linux/man-pages/man3/setsockopt.3p.html>. (accessed 5 May 2022).
- [44] Nikolay Sivko. How ping measures network round-trip time accurately using `so_timestamping`. [Online] Available: <https://coroot.com/blog/how-to-ping>, 02.23.2022. (accessed 12 May 2022).
- [45] man7.org Michael Kerrisk. `send(2)` - linux manual page. [Online] Available: <https://man7.org/linux/man-pages/man2/sendmsg.2.html>. (accessed 5 May 2022).
- [46] man7.org Michael Kerrisk. `msg(3)` - linux manual page. [Online] Available: <https://man7.org/linux/man-pages/man3/msg.3.html>. (accessed 5 May 2022).
- [47] James F. Kurose and Keith W. Ross. *Computer networking: A top-down approach*. Always learning. Pearson Education, Harlow, 6th ed., international ed. edition, 2012.
- [48] ZYXEL. How can i calculate the switching forwarding rate and packet forwarding rate of ports. [Online] Available: <https://kb.zyxel.com/KB/searchArticle!gwsViewDetail.action?articleOid=007011&lang=ENg>. (accessed 13 May 2022).
- [49] RTDS. *RSCAD Controls Library Manual*. RSCAD v5.011 (accessed 24 May 2022).



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Ich möchte mich sehr herzlich bei Technischen Universität Wien und der Florida State University (FSU) Center for Advanced Power Systems (CAPS) für diese kooperative Diplomarbeit bedanken. Im Zuge des Studiums und vieler Arbeitsstunden in den Lernräumen des Zentralen Informationsteams (ZID) hat sich über die Jahre eine sehr vertraute Lerngruppe etabliert. Die somit selbst ernannte "ZID-Raum-Gan" gehört mittlerweile nicht nur zu meinem engsten Freundeskreis, sondern auch zu sehr wertgeschätzten Arbeitskollegen die immer für eine wissenschaftliche Diskussionsrunde zu begeistern waren. Ich möchte mich auch bei meinem Vater Michael Reisinger bedanken, durch dessen Unterstützung über die Jahre hinweg, ich den Fokus auf mein Studium lenken konnte. Egal wie ermüdend und anstrengend das Studium auch teilweise war, gab es vor allem eine Person die immer hinter mir stand und auf dessen emotionale Unterstützung ich mich immer verlassen konnte, danke Marlies Metzich. Ein großer Dank geht auch an meine Betreuer Wilfried Steiner, Mark Stanovich und Michael (Mischa) Steurer die mich während meiner Diplomarbeit betreut haben. Zu guter Letzt bedanke ich mich bei allen namentlich nicht erwähnten Freunden und Familienmitgliedern die immer hinter mir standen und mich während meiner Studiums Zeit unterstützt haben.



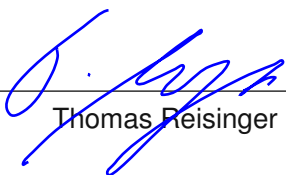
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Thomas Reisinger, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Juni 2022


Thomas Reisinger



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.